PB-239 898

ROUTING OF SOLID WASTE COLLECTION VEHICLES

APPENDIX A: MANUAL FOR USE OF THE COMPUTER CODES

ILLINOIS UNIVERSITY

PREPARED FOR
NATIONAL ENVIRONMENTAL RESEARCH CENTER

DECEMBER 1974

DISTRIBUTED BY:

NTIS
National Technical Information Service
U. S. DEPARTMENT OF COMMERCE

TECHNICAL REPORT DATA
(Please read Instructions on the reverse before completing)

1. REPORT NO. EPA-670/2-74-036a	2.
3. PB 239 898	
4. TITLE AND SUBTITLE: ROUTING OF SOLID WASTE COLLECTION VEHICLES; Final Report and Appendix A: Manual for Use of the Computer Codes.	5. REPORT DATE: December 1974-Issuing Date
	6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S): Jon C. Liebman	8. PERFORMING ORGANIZATION REPORT NO.
9. PERFORMING ORGANIZATION NAME AND ADDRESS: University of Illinois, Urbana, Illinois 61801	10. PROGRAM ELEMENT NO.: 1DB314;ROAP 02AAE;TASK 04
	11. CONTRACT/GRANT NO.: R801289
12. SPONSORING AGENCY NAME AND ADDRESS: National Environmental Research Center, Office of Research and Development, U.S. Environmental Protection Agency, Cincinnati, Ohio 45268	13. TYPE OF REPORT AND PERIOD COVERED: Final
	14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

See EPA-670/2-74-035a, EPA-670/2-74-035b, and EPA-670/2-74-036b

16. ABSTRACT

This report describes research on computer-based and manual techniques for the routing of solid waste collection vehicles. Algorithms are given for the routing of single vehicles under three conditions: (1) all streets are two way, (2) all streets are one way, and (3) some streets are two way and others are one way. In addition, a heuristic algorithm is provided for routing several vehicles in a larger area. Appendix A is a manual for use of the computer codes. Appendix B (separately bound) is a heuristic solution to the m-postmen's solution.

17. KEY WORDS AND DOCUMENT ANALYSIS

a. DESCRIPTORS	b. IDENTIFIERS/OPEN ENDED TERMS	c. COSATI Field/Group
*Collection, *Routing, Refuse, *Computer programming, Management, Computers, *Vehicles, Mathematical models, Algorithms, Heuristic methods	Solid waste, Solid waste management, *Solid waste collection, *Computing routing, Heuristic routing	13B

18. DISTRIBUTION STATEMENT: Release to public	19. SECURITY CLASS (This Report): UNCLASSIFIED	21. NO. OF PAGES: 67
	20. SECURITY CLASS (This page): UNCLASSIFIED	22. PRICE: 4.25

EPA Form 2220-1 (9-73)

Reproduced by
NATIONAL TECHNICAL INFORMATION SERVICE
US Department of Commerce
Springfield, VA. 22151

EPA-670/2-74-036a
December 1974

071058

ROUTING OF SOLID WASTE COLLECTION VEHICLES

Final Report

and

Appendix A: Manual for Use of the Computer Codes

By

Jon C. Liebman
University of Illinois
Urbana, Illinois 61801

Program Element No. 1DB314

Project Officer

Donald A. Oberacker
Solid and Hazardous Waste Research Laboratory
National Environmental Research Center
Cincinnati, Ohio 45268

NATIONAL ENVIRONMENTAL RESEARCH CENTER
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
CINCINNATI, OHIO 45268

ia

REVIEW NOTICE

The National Environmental Research Center--Cincinnati has reviewed this report and approved its publication. Approval does not signify that the contents necessarily reflect the views and policies of the U.S. Environmental Protection Agency, nor does mention of trade names or commercial products constitute endorsement or recommendation for use

FOREWORD

Man and his environment must be protected from the adverse effects of pesticides, radiation, noise and other forms of pollution, and the unwise management of solid waste. Efforts to protect the environment require a focus that recognizes the interplay between the components of our physical environment--air, water, and land. The National Environmental Research Centers provide this multidisciplinary focus through programs engaged in

- studies on the effects of environmental contaminants on man and the biosphere, and
- a search for ways to prevent contamination and to recycle valuable resources.

As part of these activities, this report describes research on computer based and manual techniques for the routing of solid waste collection vehicles.

A. W. Breidenbach, Ph.D.
Director
National Environmental
Research Center, Cincinnati

ABSTRACT

This final report describes research on computer-based and manual techniques for the routing of solid waste collection vehicles. Algorithms are given for routing of single vehicles under 3 conditions:

a) All streets are two-way

b) All streets are one-way

c) Some streets are two-way and others are one-way

In addition, a heuristic algorithm is provided for routing several vehicles in a larger area.

ACKNOWLEDGEMENTS

In addition to the author of this report, the research team in this project consisted of Dr. Magne Watine, Dr. Saman Hong, and Dr. James Male. Each of these researchers contributed an important portion of the results as his doctoral research.

There are many other persons who have made significant contributions. Dr. Mandell Bellmore was associated with the project in its first year, and suggested many of the theoretical developments incorporated in the mixed street direction algorithm. Mr. Lewis Rossman gave generously of his time in serving as a "sounding board", and provided considerable programming expertise. The k-match computer code was provided by Dr. Lee J. White. Several real street networks were provided (and our results with them analyzed) by Mr. David Reuben.

In addition, discussions with several students, professors, and others led to improvements and advances in our methods. We are particularly grateful to Dr. Geoff Berlin, Dr. Jack Elzinga, Mr. Dan Hogenauer, Dr. Judith Liebman, Dr. David Marks, Dr. Charles ReVelle, Mr. Dan Sheer, and Mr. Makram Suidan.

CONTENTS

	Page
Chapter 1: Introduction.....	1
Chapter 2: Review of Some Fundamental Results.....	5
2.1 Euler's Problem.....	5
2.2 Kwan and the Chinese Postman.....	9
2.3 The Shortest Path Concept.....	12
2.4 The Matching Solution.....	18
2.5 Properties of an Optimal Tour.....	21
Chapter 3: Single Vehicle Tours.....	23
3.1 Single Vehicle Tours on Undirected Graphs.	23
3.2 Single Vehicle Tours on Directed Graphs...	29
3.3 Single Vehicle Tours in Mixed Graphs.....	36
Chapter 4: Multiple Vehicle Tours.....	44
Chapter 5: Summary and Discussion.....	50
References	52
Appendix A: Manual for Use of the Computer Codes.....	53
A.1 Data Collection.....	53
A.2 TOUR1W.....	54
A.3 TOUR2W.....	55
A.4 TOURM.....	56
A.5 MPOST.....	58
Appendix B:	Separately Bound

LIST OF FIGURES

		Page
Figure 1:	The Konigsberg Bridge Problem.....	6
Figure 2:	Exchange of Retraced Edges in a Cycle.....	11
Figure 3:	Paths of Retracings.....	13
Figure 4:	Elimination of Multiple Retracings.....	15
Figure 5:	Cycle Formed by Retracings and Shortest Path.....	16
Figure 6:	Two Sets of Shortest Path Retracings.....	17
Figure 7:	Graph of an Actual Street Network.....	19
Figure 8:	Original Graph and Derived Graph of Odd-Degree Nodes.....	20
Figure 9:	Elimination of A U-Turn.....	26
Figure 10:	Representation of a Directed Case with Depot.....	32
Figure 11:	A Spanning Arborescence on a Unicursal Graph.....	35
Figure 12:	Unicursality Conditions on Nodes in a Mixed Graph.....	38
Figure 13:	A Mixed Graph Which is Not Unicursal.....	39
Figure 14:	The Graph of Fig. 13 Made Unicursal and Symmetric.....	41
Figure 15:	Districting on a Unicursal Graph.....	46

CHAPTER 1: INTRODUCTION

The research reported herein deals with various mathematical and computer-based techniques for determining routes for solid waste collection vehicles. Four complete algorithms are discussed: three for determining routes for an individual truck under various circumstances, and one for determining routes for several trucks. This report is intended for reading by those who may wish to use the techniques discussed; the complex mathematical details have been omitted from the body of the report, and are included in appendices to the first annual report and to this report.

Although the principal intent of this report is to explain the development and use of the computer algorithms, a great deal of insight into the construction of good routes is also provided. This insight should prove useful to those who may not have access to the large computers needed for solution of the algorithms and who need to construct routes without the use of computers.

The overall problem considered is that of the Director of Public Works of some municipality, who needs to provide solid waste collection service to the residents at minimum or near-minimum cost. We are concerned herein with how such a municipality may be divided into collection districts, and how a collection vehicle may be routed within a district. The objective is to provide the required service with a minimum of excess travel by the collection vehicle.

At the outset, a number of problems which have not been considered in this research should be noted. A major problem in the design of any solid waste collection system is the determination of the appropriate quality of service. Quality encompasses such parameters as frequency of collection, location of pickup (curbside, back yard, etc.), types of containers to be used, varieties of separation to be required, types of vehicles to be used, etc. These measures are taken as given in the development of the techniques which follow. This is not to say that decisions regarding quality are considered to be easy ones; they are simply decisions to be made externally to the framework of this research. It is true, however, that the algorithms developed herein will permit a decision-maker to explore the routing cost implications of various quality decisions, and thus provide added information in selecting among alternative plans.

A second area of major concern in most collection systems is the question of equity among the workers. Every solid waste system manager contacted in the progress of this research felt that he needed a better, more quantitative method for determining a fair

day's work, and for balancing the workloads between his collection crews. The definition of a fair day's work is clearly outside the scope of this research, and remains as a major problem. The balancing of workloads between crews is, to some extent, addressed by this research, although the major consideration is the solid waste load on the collection vehicle and not the equalization of time among crews.

It should also be pointed out that, although solid waste generation is stochastic in nature, it is treated deterministically in this research. We assume that the amount of solid waste generated on any street segment is known in advance, and not subject to random variation. We also assume that the allowable load on a collection vehicle is fixed and permits no occasional violation. Neither of these assumptions is, of course, true. On the other hand, it is common practice to assign specified areas to be collected by single trucks, and the drivers of these trucks commonly follow identical routes each time they collect. Thus, it appears that there is sufficient flexibility in vehicle loading to permit the random variations in amounts to be collected to be accommodated without rerouting the vehicles.

A precise definition of the costs which are considered in this research will aid in an understanding of the overall problem. When a collection vehicle travels over a particular street segment, the vehicle may be involved in performing collection or it may simply be traversing the street to arrive at a point where it will perform collection. An important principle is that the total amount of collection travel which must be done is invariant; the streets on which collection must be done are predetermined, and the total cost of doing such collection (in manpower and vehicle travel) is fixed. Thus, the only costs which are manipulatable by routing are the costs associated with the traversing of streets without collecting. The costs incurred by this "idle" travel are two: the manpower costs (which are usually based on time), and the vehicle costs (which are usually based on distance). We may associate with any street segment a traverse cost, which is the sum of these two costs and represents the cost of travelling over the segment without collecting. The traverse cost may be computed by measuring the distance of the segment and calculating the distance cost for the vehicle, and then adding to this an estimated cost of the manpower-time involved in making the traverse. For most cases, an average vehicle speed in traversing may be assumed, thus making traverse cost directly proportional to distance. Since much of the fundamental research underlying this study has been aimed at minimizing distance or length of travel, the reader should keep in mind that one may always substitute traverse cost for length, calculating traverse cost as described above or in any other way which seems appropriate.

There are some other costs associated with a vehicle route which

are not specifically considered in the algorithms which follow. These are principally intersection costs, associated with the time it takes to make a left turn, or to tur.. the collection vehicle around. The minimization of these costs will be discussed in later sections.

The problem which this research addresses may now be precisely stated. We are given an area in which solid waste is to be collected. The area is defined by a set of street segments (called links), and a set of intersections (called nodes). Associated with each link is a known load of solid waste to be collected, and a traverse cost. (It should again be emphasized that the traverse cost reflects cost of travel without collection; the cost of collecting on the street segment is unavoidable). Some links may have no load at all; these are links which may be traversed but on which no collection is required. We are also given a capacity of the collection vehicles, and the location of the garage in which vehicles are housed and the depot to which vehicles deliver their collected waste. The overall problem is now three-fold:

a) To divide the area into districts, which are defined to be contiguous areas each of which may be collected by a single vehicle without exceeding its capacity and without returning to the depot before completing the district.

b) Within each district, to determine a tour, which is a complete specification of the order in which links are to be collected and traversed, and includes the origin (at garage or depot) and the termination (at the depot).

c) To aggregate the tours into routes, each containing one or more tours. A route is defined to be a day's work for a vehicle and its associated crew.

The objective is considered to be the minimization of total traverse cost, including the trips to and from garage and depot.

As will be seen in later sections, it is easiest to attack sub-problem b) first. That is, we take the district as given, and attempt to determine a single tour within the district which minimizes traverse cost. We are able to present optimal algorithms for determining such tours. We then present a heuristic for solving sub-problems a) and b) jointly. This algorithm is not optimal, but our experience has indicated that it determines rather good districts and tours at a very low cost in computation time. No algorithm is presented for the solution of sub-problem c), which involves equity among crews, and which we believe is better done by man than by machine.

It is of particular importance to note that our algorithms for sub-problems a) and b) do not, in fact, determine unique tours.

Rather, they specify the links which must be traversed, permitting a tour designer to select from a large number of possible tours, all with minimum traverse cost. A tour-building algorithm is included in our computer codes, so that one possible tour will always be provided. In addition, we discuss techniques for building tours by hand which will aid in finding other optimal tours. This flexibility permits the decision-maker to take into consideration other factors in the design of the collection system which are not explicitly included in our techniques. In this light, we emphasize that all of the algorithms and methods presented herein should be considered as tools which provide information, rather than as ways of finding the answer. It is our belief that this is the proper role of mathematical models and optimization methods in complex public decision problems.

The obvious application of the techniques presented in this report is for the redesign of an entire municipal collection system. This is not, however, the only use of the methods, and it may, in fact, be the least important use. The algorithms for constructing a tour within a given district may be used to make minor changes in a single vehicle's route, and the districting algorithm may be used for the construction of a single vehicle's route which consists of two or three tours. By far the major cost in using the algorithms is the accumulation of the data (principally traverse costs) which are required. Once such data are accumulated, small-scale applications such as the above become particularly appealing. It is even conceivable that collection system managers will be able to use time-sharing computer systems with large pre-stored data bases to re-route vehicles on a daily or weekly basis to avoid construction or other interruptions of traffic. In most such uses, the principal advantage of the computer-based algorithms will not be their optimality; rather, it will be the speed and ease with which information can be obtained, providing rapid re-routing where it cannot be done at present.

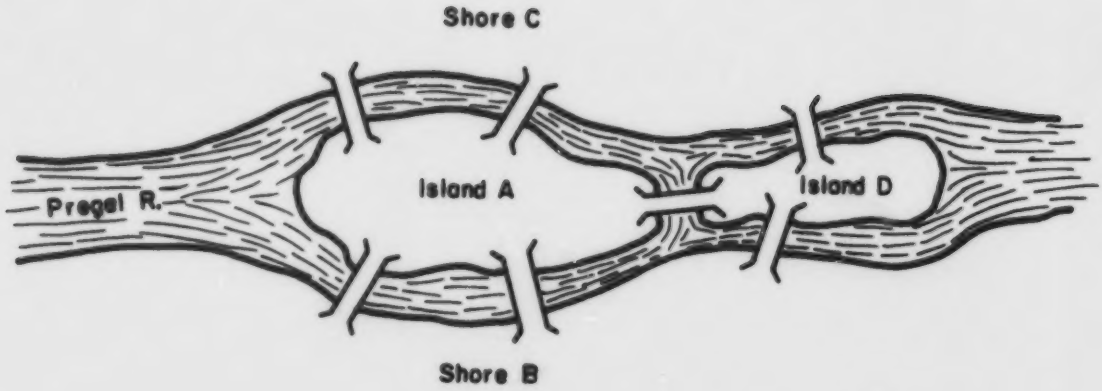
CHAPTER 2: REVIEW OF SOME FUNDAMENTAL RESULTS

Much of the fundamental research on which the current study is based has been done by mathematicians in the field of graph theory, some of it more than two-hundred years ago. This chapter reviews the results of their work which are directly applicable in the techniques which are presented later. It should be pointed out that there are other approaches to the routing problem than those used in this research; no attempt has been made to review alternate approaches or the results which underlie them. The emphasis herein is on the insight provided by work which has a direct bearing on our algorithms, not on historical or academic interest. Although some of the ensuing discussion may seem to be very theoretical, the reader is urged to master it for the understanding it provides.

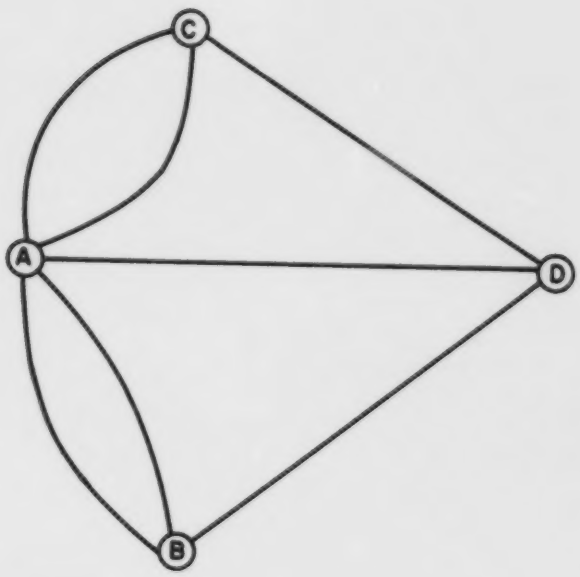
2.1 Euler's Problem

Leonhard Euler (1707-1782) became the father of modern graph theory and at the same time provided the most important fundamental result bearing on the collection vehicle routine problem while studying the now-famous Konigsberg Bridge problem [Euler, 1736]. There were two islands in the Pregel River, linked to each other and to the banks of the river by seven bridges, as shown in Figure 1a. A popular pastime was to try and find a way to start at any one of the four land areas, cross each bridge exactly one time, and return to the starting point. In attempting to solve the problem, Euler replaced the four land areas with four nodes, which are simply the intersections of bridges and represented each bridge by a link. The resulting diagram (Figure 1b), now called a graph, represents all of the essential features of the original problem. The translated problem is to start at any one of the nodes, travel on each link exactly once, and return to the starting node. The reader will recognize that the corresponding collection problem is to start at one of the intersections (nodes), collect on each of the streets (links), and return to the starting node without traversing any street (travelling over it a second time, without collecting). If this can be done, it represents a minimum cost tour over the district consisting of these seven streets and four intersections, since there is no traverse cost at all. The only cost is collection cost, which is fixed.

Euler then generalized the problem by deriving conditions which are necessary and sufficient in any graph for there to be a tour which travels on every link exactly once and returns to its starting point. The two conditions are: a) the graph must be connected and b) every node must be of even degree.



(a)



(b)

Figure 1: The Konigsberg Bridge Problem (Adapted from Harary, 1969)

A connected graph is one in which it is possible, by travelling over links of the graph, to get from any node to any other node.[1] It is obvious that if a graph is not connected there is no possible tour on all of its links.[2]

The degree of a node is the number of links incident upon the node. The necessity for every node to be of even degree is made intuitively clear by considering the problem in the following manner. Every node (except the one at which we start and finish) is first entered and then left. We may enter and leave the node as many times as necessary, but each time we do so we cover two of the links incident on the node. If any node has an odd number of links incident upon it (i.e., is of odd degree), then we are ultimately faced with a link on which to enter and no corresponding link on which to leave. Similar reasoning leads to the conclusion that the starting (and finishing) node must also be of even degree, since we first leave the node, then enter and leave it a number of times, and finally enter it to terminate the tour. Clearly, then, if one or more nodes is of odd degree a tour over all links exactly once is not possible.

From the above we see that it is necessary to have connectedness and all nodes of even degree in order to be able to construct a tour which travels over every link exactly once and returns to its starting point. The sufficiency of these conditions (that is, a proof that every graph which satisfies these conditions has such a tour) is not immediately obvious. The proof is sketched here; for more detail the reader is referred to Harary [1969]. We first show that every connected graph of even degree consists of a set of cycles, and then that the tour which we require can be constructed on any graph which consists of a set of cycles.

A cycle is any tour which travels over a set of links of the graph and returns to its starting point; it need not travel over all links nor visit all nodes. In any connected graph of even degree, we may identify any cycle and remove all of the links which make up the cycle; the remaining graph is still of even degree because we have removed either zero or two links incident on each node. We may therefore continue in this fashion until all links have been removed from the graph, which proves that the graph consists of a set of cycles.

To demonstrate that we can construct the required tour on a graph

---

[1]This definition is not correct when some of the links of the graph have a direction associated with them. Precise definitions may be found in Harary [1969].

[2]The exception to this statement is a graph whose disconnected portions consist of nodes with no links.

which consists of a set of cycles, we first consider any cycle of the graph. If that cycle is the entire graph, then it is the required tour, and we are finished. If it is not the entire graph, then there is at least one other cycle which touches our first cycle at a node. We may construct a new, larger tour by breaking our first cycle at the common node, inserting the second cycle (which starts and finishes at the common node), and then completing the first cycle. We may continue in this fashion until we have used every cycle, and thus travelled over every link. This proves that the required tour may be constructed on any connected graph in which all nodes are of even degree, and provides a method for constructing such a tour.

Two further results of Euler's study are worthy of note. First, every graph (and therefore every street network) must have an even number of nodes of odd degree. This conclusion is based on the total number of incidences in the entire graph. Every link touches two nodes, one at each end. Thus, the total number of incidences of links on nodes is exactly two times the total number of links; there is an even number of incidences. The sum of the degrees of all nodes must equal the total number of incidences and therefore must be even. This is clearly true if all nodes are of even degree. If some nodes are of even degree and some of odd degree, the sum of the degrees of the even nodes is obviously an even number, so the sum of the degrees of the odd nodes must also be an even number. In order to sum a group of odd numbers and get an even number, the number of odd numbers must be even. Thus, there is an even number of nodes of odd degree.

A second conclusion is of particular importance in finding tours which travel over every link, but do not start and finish at the same node. If there are exactly two nodes of odd degree, and all other nodes are of even degree, then there exists such a tour, starting at one of the nodes of odd degree and finishing at the other. Similarly, if there are exactly four nodes of odd degree, then two tours may be constructed, which, between them, travel over every link, each starting at one of the four nodes and finishing at another. There is clearly more than one set of such tours, depending on which pairs of odd nodes we choose to place in each tour. The same reasoning may be extended to determine how many tours are necessary for any graph, with any (even) number of nodes of odd degree.

Because of Euler's pioneering work in this area, tours which travel over every link in a network exactly once, and return to their starting point, are called euler tours. A connected graph which has every node of even degree (and, therefore, has an euler tour) is called a unicursal graph. It is clear that the graph of the Konigsberg Birdges is not unicursal, so no euler tour exists.

Before continuing with additional results, it is important to note a limitation of the theory. The results which we have discussed thus far, and most of those which follow in this chapter, are restricted to graphs in which the links have no associated direction (i.e., two-way streets). In graph theory terminology, links which are undirected are called edges; links which have an associated direction (such as one-way streets) are called arcs, and the word 'link' is used to apply to either type. More properly, the preceeding paragraphs should have discussed edges rather than links. We will defer a discussion of graphs containing arcs until later in this report.

It should be noted that Euler was in no way concerned with the construction of a tour which travelled over every edge at least once, if this required travelling some edges more than once. Rather, his interest was in characterizing graphs in which it was possible to travel over every edge exactly once. If we assume that every edge in the graph represents a street segment on which collection must take place, then the problem which we wish to solve involves travelling over every edge at least once, and minimizing the length (traverse cost) of the extra travel which we must do on edges which have already been collected.

An important way of viewing the problem results from the fact that, if the graph of our street network is not unicursal, any tour which travels on each street at least once will also travel on some streets more than once. If we draw the "retracings" on the graph as additional edges, the resulting graph must be unicursal, since we are able to construct an euler tour on it. Our problem, then, is to begin with a graph which is not unicursal, and to add (duplicate) edges to it so that it is unicursal, in such a way that the length (traverse cost) of the added edges is minimum. Euler's result then guarantees that we will be able to determine a tour on the new graph which travels over every edge (including the duplicates) exactly once and returns to the starting point. Since the traverse cost of the added edges is minimum, the traverse cost of our tour is minimum. Construction of a unicursal graph in this manner implies, incidentally, that the duplicated edges must be determined in such a way as to make the nodes of odd degree into nodes of even degree, while retaining the even-degree property of the other nodes.

2.2 Kwan and the Chinese Postman

The first published attempt to solve this minimization problem was by Kwan Mei-Ko [1962] who was interested in determining postman's tours. The postman's problem is exactly analogous to the collection problem: the edges in the graph are street segments along which mail must be delivered; and the postman wishes to

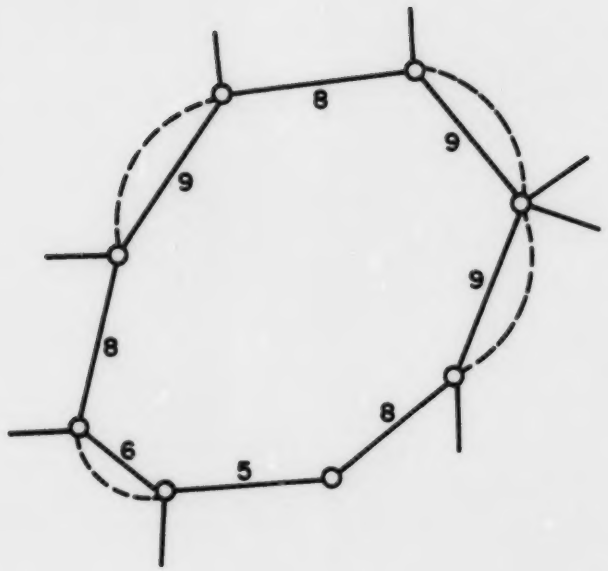
start at the post office, make his rounds, and return. Kwan recognized that any complete tour, whether of minimum distance or not, could be represented as an euler tour on a unicursal graph with duplicated edges, and presented a method for starting with any known tour and improving it until it is optimal. Fundamental to his technique is the idea (discussed earlier) that a unicursal graph consists of a set of cycles. If we examine any cycle in the unicursal graph of a complete postman's tour, we find that it consists of some edges which are not retraced and (possibly) other edges which are retraced. Kwan's contribution was in discovering that if, in such a cycle, the sum of the lengths of the retraced edges is greater than the sum of the lengths of the non-retraced edges, then the graph may be easily altered to reduce the length of the retracings while retaining the unicursal property. In Figure 2a, we show a single cycle removed from a unicursal graph, including the retraced edges (dashed lines). The lengths of the edges are shown beside the edges. The sum of the lengths of the non-retraced edges is 29, while the sum of the lengths of the retraced edges is 33. Note that, including retracings and edges not in the cycle, every node is of even degree, as is required for the graph to be unicursal. Figure 2b shows the same cycle after the retraced edges and non-retraced edges have been exchanged. Every node remains of even degree, and thus the graph remains unicursal. However, the length of retracings in the cycle has been reduced from 33 to 29. Thus, this exchange has reduced the traverse cost by 4. Kwan presents a proof that when there exists no cycle in the graph which contains a greater length of retraced edges than non-retraced edges, any euler tour which is constructed on the graph contains a minimum total traverse cost. An exchange of retraced edges for non-retraced edges can always be made in any cycle which does not satisfy the above condition. Thus, Kwan's result can be placed in the form of a simple algorithm:

1) Begin with a unicursal graph, containing some retracings. This graph represents any tour (such as the one the postman currently follows).

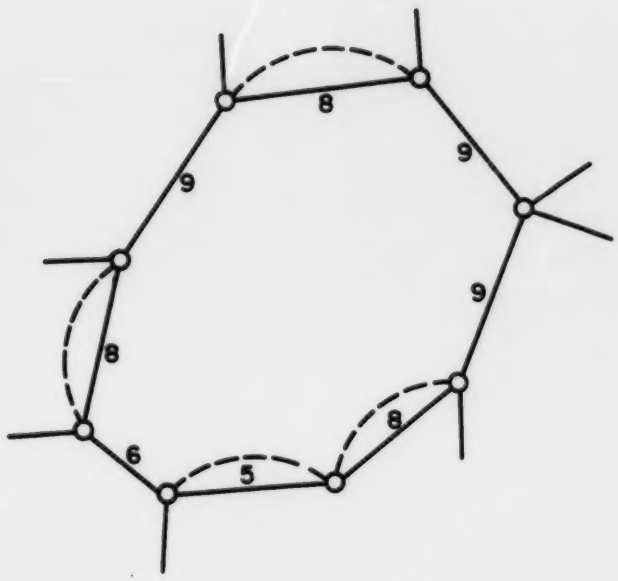
2) Examine any cycle in the graph. If the sum of lengths of retraced edges in the cycle is greater than the sum of lengths of non-retraced edges, remove all of the retracings and add retracings of all of the non-retraced edges.

3) Continue in this fashion until no further exchanges can be made.

4) Construct a tour on the resulting unicursal graph. The method of construction is given in the outline of the proof of sufficiency for unicursality: begin with any cycle and insert additional cycles to make a larger and larger tour until the entire graph is covered.



(a)



(b)

Figure 2: Exchange of Retraced Edges in a Cycle

Kwan's algorithm has one serious drawback. In any realistic street network, the number of cycles is likely to be extremely large. Thus, the inspection of all cycles to insure that there are no further exchanges will be very time-consuming, and makes the algorithm impractical even with the use of high-speed computers. However, Kwan's work is quite useful for the improvement of existing tours on an ad hoc basis, when optimality is not sought or the computer algorithms discussed later in this report are not available. A tour designer who keeps in mind the potential of making improvements by exchanging retracings in cycles with an excess of retraced lengths may frequently be able to identify such improvements, without conducting the exhaustive search necessary to guarantee optimality.

Because of Kwan's work, the problem of finding a minimum-length edge-covering tour in a connected, undirected graph has come to be called The Chinese Postman's Problem.

2.3 The Shortest Path Concept

Another way of looking at the Chinese Postman's problem is provided by the work of Glover [1967] and of Edmonds [1965]. In a graph which is not unicursal, what is required is the duplication of appropriate edges so as to make the graph unicursal. As previously suggested, this implies converting every odd-degree node into one of even degree, while retaining the even-degree property of the remaining nodes. Clearly, when we duplicate a single edge between two nodes, we change the degree of both nodes. If both nodes are originally of odd degree, they both become even-degree nodes. But if one node is of odd degree and the other of even degree, duplicating the edge between them simply reverses the odd and even property. The node which was of even degree is now of odd degree, and we must now duplicate another edge touching this node to make it even once again. Such a process will have to continue until we duplicate an edge terminating at a second node which was originally of odd degree, thus constructing a path of duplicated edges between two nodes of odd degree. Figure 3a shows such a path, beginning and ending at nodes which were originally of odd degree, and passing through several nodes which were originally of even degree. The only way in which a path of duplicated edges can begin and end at nodes which are originally of even degree is if the entire path is duplicated an additional time, as shown in Figure 3b, restoring the even-degree property. Clearly, such duplications are not necessary, since without any of the duplicate edges the nodes are of even degree. Thus we are led to the conclusion that in any "good" construction of an euler tour, all of the duplicated edges or retracings form paths which begin and end at nodes originally of odd degree, and every originally odd-degree node is at the end of one such path.

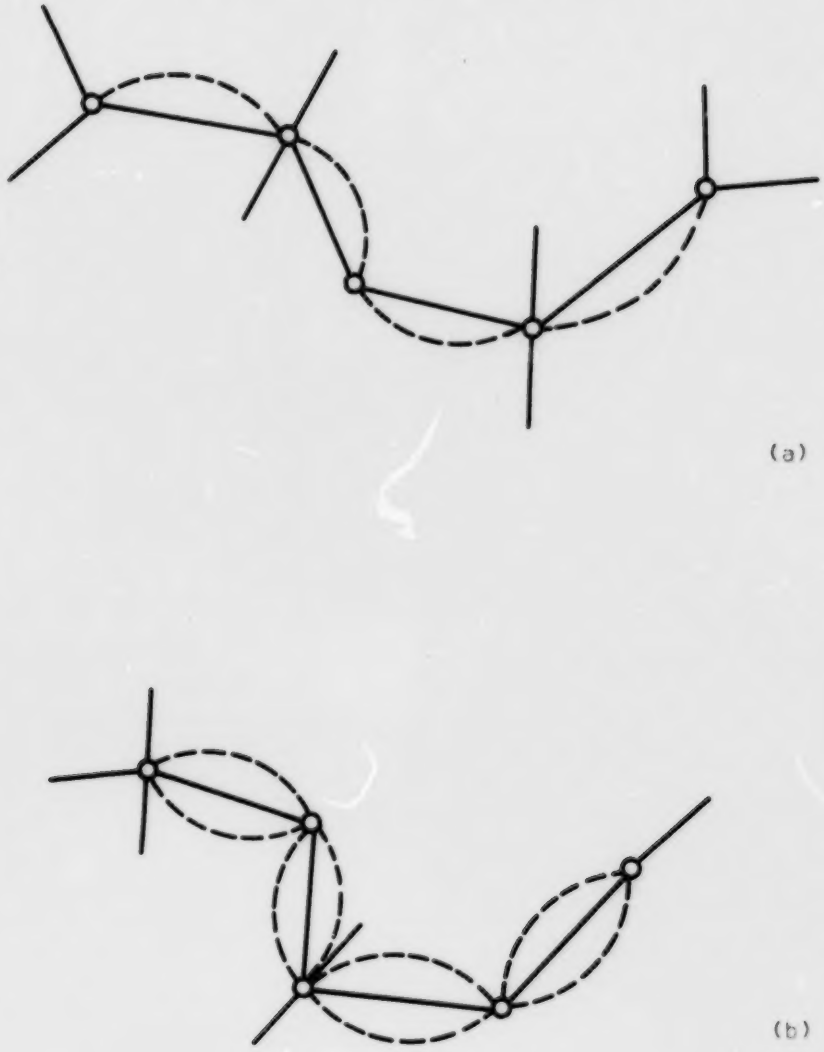


Figure 3: Paths of Retracings

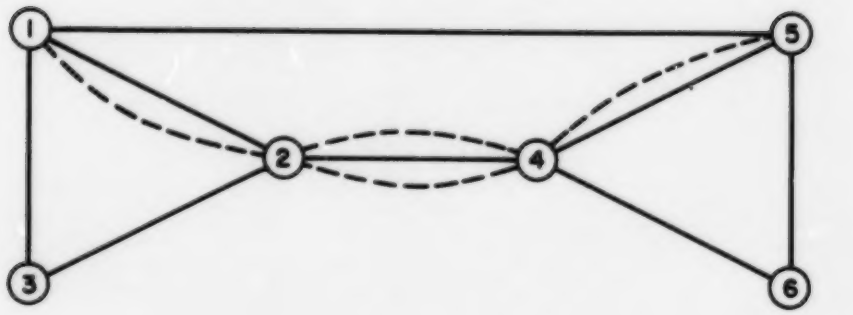
One additional property of good tours can be discovered by inspection of Figure 4. In Figure 4a, there are four nodes of odd degree. One path of duplicated edges connects node 1 with node 4, passing through node 2; the other connects node 2 with node 5, passing through node 4. As a result, the edge between nodes 2 and 4 is duplicated twice. An obviously better solution is shown in Figure 4b, where the two duplications of the edge between nodes 2 and 4 have both been eliminated, and the duplicate paths now connect node 1 with node 2 and node 4 with node 5. This result may be generalized in the following way. Any time an edge is duplicated twice (or any other even number of times), all of the duplicates of the edge may be removed and the graph will remain unicursal. Any time an edge is duplicated an odd number of times, all of the duplicates of the edge except one may be removed and the graph will remain unicursal. Thus, no edge need be duplicated more than once.[3] The resulting unicursal graph still retains the property that all duplicated edges form paths which begin and end at nodes of odd degree.

The above reasoning immediately leads us to an additional useful result, shown in Figure 5. If a path of duplicated edges between two nodes of odd degree is not the shortest available path between those nodes, we can improve the traverse cost in the graph by eliminating the duplicated edges and replacing them with duplicated edges in the shorter path. Thus, in any optimal tour, it must be true that the paths of duplicated edges must be shortest paths connecting the odd nodes at each end. This is merely a restatement of one portion of Kwan's cycle property, since the two paths form a cycle with a greater length of retraced edges than non-retraced edges. Nodes 1 and 6 are originally of odd degree, and the path from node 1 to node 6 via node 7 is clearly shorter than that which is retraced.

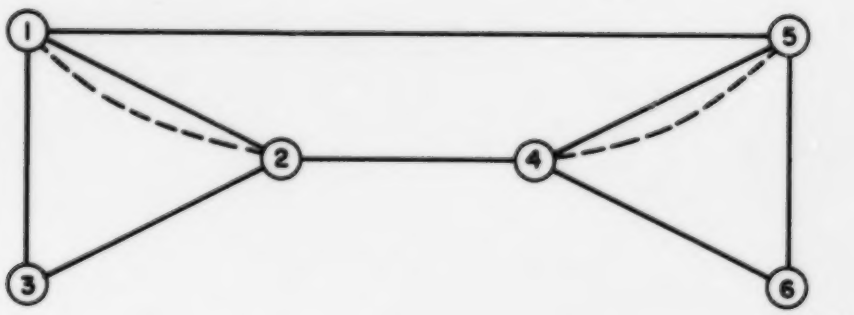
It is not sufficient for optimality, however, to have every path of duplicated edges form a shortest path between two nodes of odd degree. Optimality also depends on which pairs of odd-degree nodes we choose to connect. This is demonstrated by Figure 6, where the only nodes of odd degree are nodes 1,4,5 and 8. The duplicated edges in Figure 6a are the shortest paths, respectively, connecting node 1 with node 4 and node 5 with node 8. However, using Kwan's result on the cycle through nodes 1,2,3,4,8,7,6,5, and back to 1 gives the improved result of Figure 6b, which connects node 1 with node 5 and node 4 with node 8. Thus, we must not only connect odd-degree nodes with shortest paths of duplicate edges, but we must also select the "right" pairs of odd-degree nodes to connect.

---

[3]It is again emphasized that we are considering only undirected graphs. This result does not hold for graphs which have some directed links (arcs).



(a)



(b)

Figure 4: Elimination of Multiple Retracings

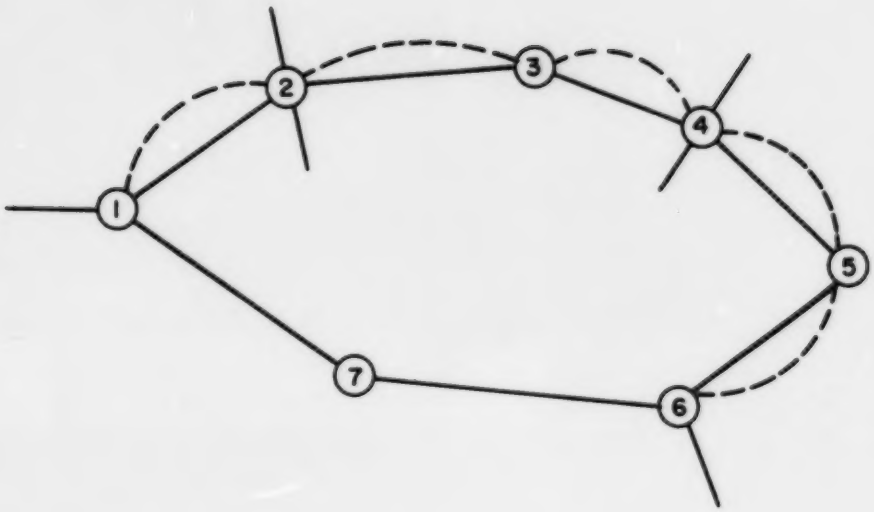


Figure 5: Cycle Formed by Retracings and Shortest Path

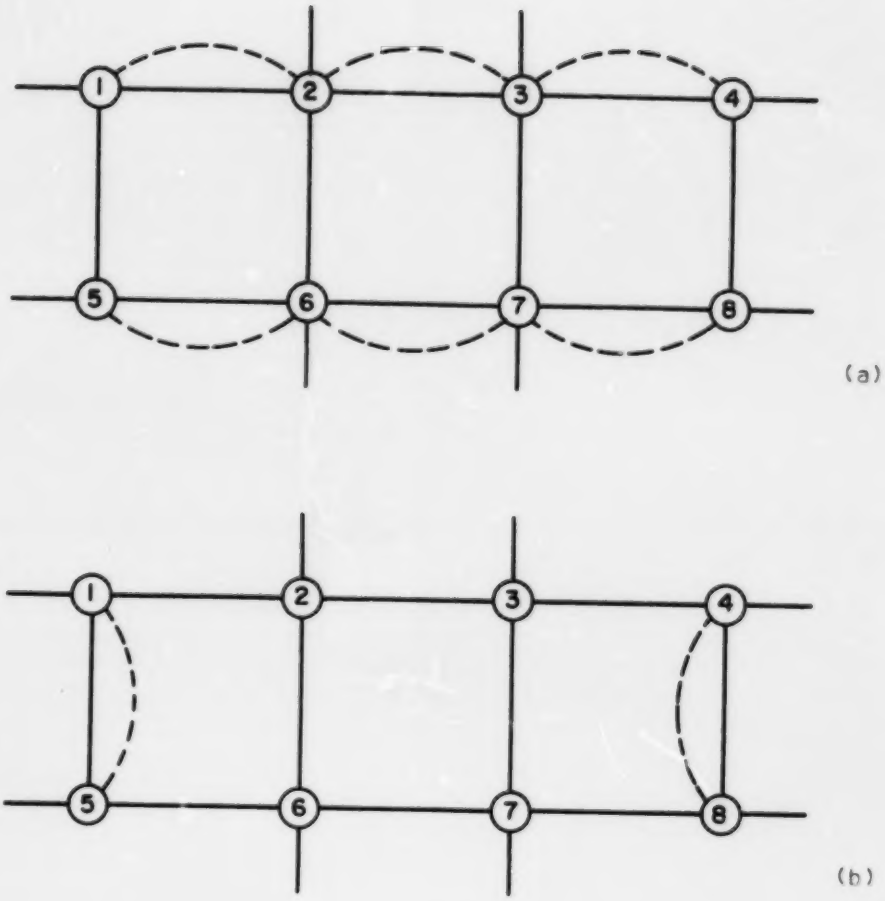


Figure 6: Two Sets of Shortest Path Retracings

It is this last point which is the crux of the mathematical difficulty of the problem. It is usually relatively easy to find the shortest path between two specified nodes by inspection, if the two nodes are fairly close together. Even when this is not easily done, there is a number of excellent and extremely efficient computer algorithms for finding shortest paths between specified nodes. However, when one is faced with the problems of "pairing up" the nodes to be connected by shortest paths, in such a way that the sum of the lengths of the paths will be minimal, intuition frequently fails. As an example, Figure 7 presents a simplified schematic of an actual street network in Indianapolis, Indiana. The edges are approximately to scale. There are 28 nodes in the graph, of which 20 (indicated by squares) are of odd degree. Thus, one must identify ten pairs of these nodes and connect the pairs by shortest paths. Even on this relatively small graph, identifying the optimal pairings is by no means simple. The optimal solution contains duplicate edges connecting the following node pairs: 2-8 (via 4), 5-6, 9-18 (via 11), 10-12, 13-19, 14-20, 15-16, 17-22, 23-26, and 24-27.

2.4 The Matching Solution

The development which finally provided an optimal solution technique for this problem is due to Edmonds [1965b]. Edmonds was interested in the graph-theoretic problem known as matching. The matching problem is to find, in any graph, a set of edges such that each node of the graph is touched by exactly one of the edges and the sum of the lengths of the edges is minimal (or maximal). He developed a very elegant algorithm for the solution of this problem. The algorithm is quite complex, and will not be described in this report.

The direct application of Edmonds' matching algorithm to the solution of the Chinese Postman's problem requires a transformation of the postman's graph. Edmonds' algorithm provides pairings of all nodes, such that the sum of the lengths of the edges connecting the pairs is minimal. The Chinese Postman's problem requires pairings of only the odd-degree nodes, which may be connected by paths, and not necessarily by single edges. The required transformation is shown in Figure 8. Figure 8a is an original postman's graph, with four nodes of odd degree. Figure 8b is a graph showing only these four nodes. The edges between the nodes in Figure 8b represent shortest paths between the odd-degree nodes in Figure 8a. Thus, for example, the edge between nodes 2 and 5 in Figure 8b represents the path from node 2 to node 1 to node 5 in Figure 8a. The solution of the matching problem on the graph of Figure 8b is the two edges shown by darker lines, connecting node 2 with node 5 and node 3 with node 8.

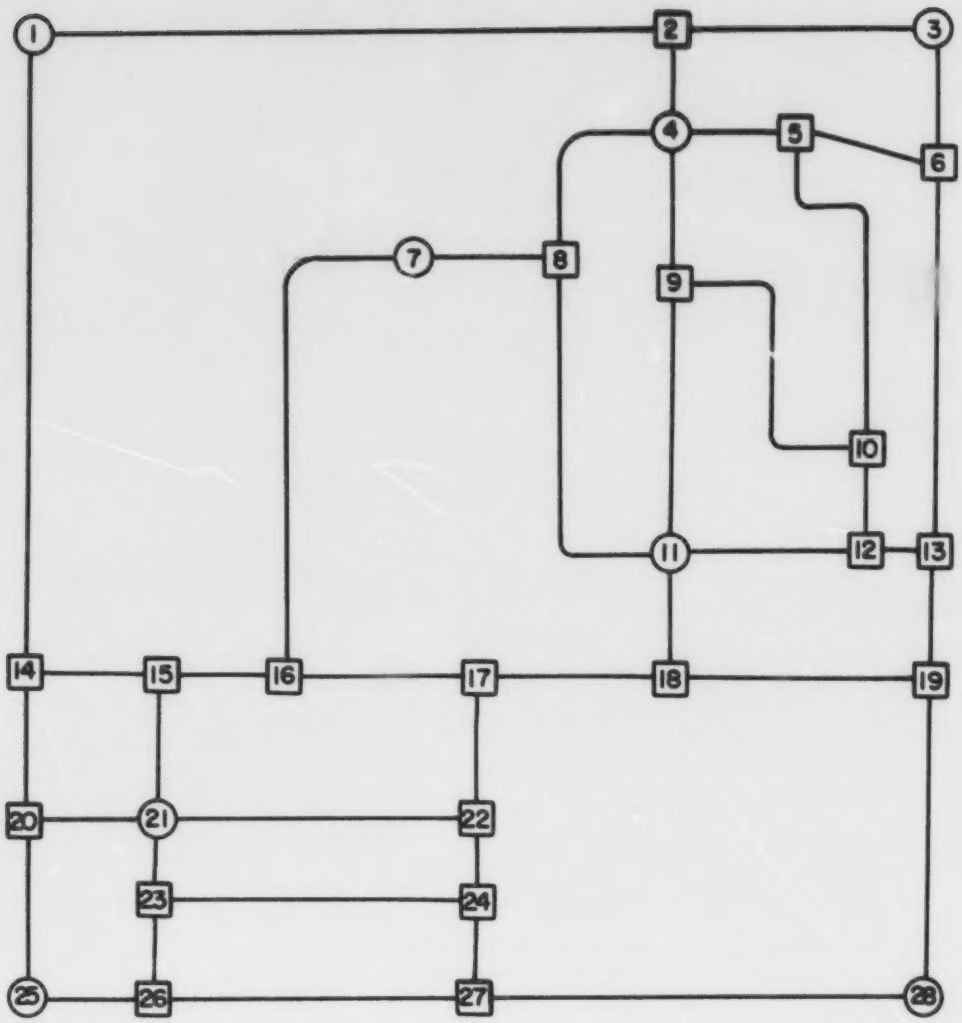


Figure 7: Graph of an Actual Street Network (Indianapolis, Indiana)

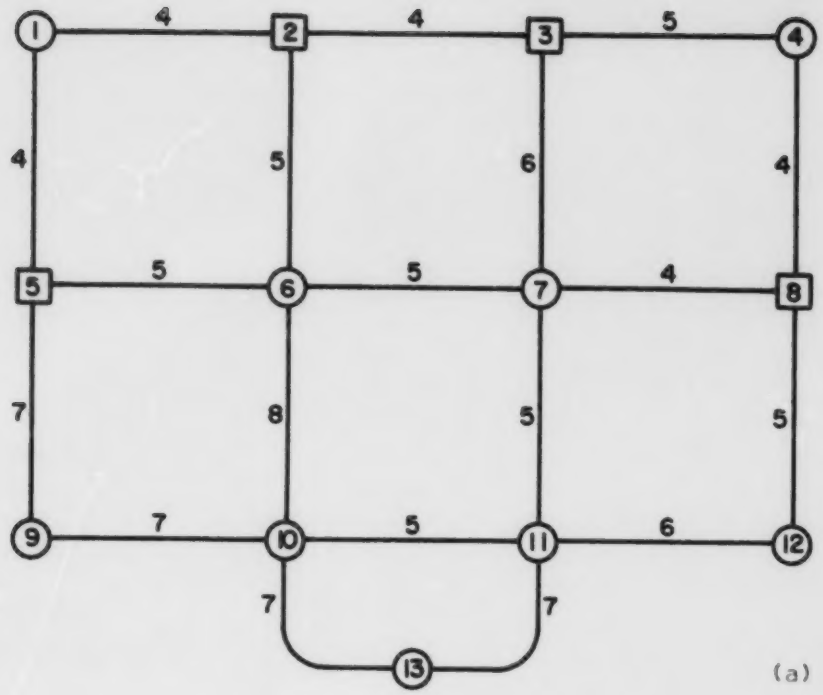


Figure 8: Original Graph and Derived Graph of Odd-Degree Nodes

This implies that the required duplicate edges in the graph of Figure 8a are the shortest paths connecting these nodes: the path from 2 to 1 to 5 and the path from 3 to 4 to 8.

The matching problem solution is thus able to provide a solution to the Chinese Postman's problem, in that it determines the construction of a unicursal graph by duplicating edges in paths connecting nodes of odd degree such that the sum of the lengths of the duplicated edges is minimal. From Euler's result, we know that an euler tour can then be constructed on the resulting graph. As a consequence, this tour will be of minimal length (minimal traverse cost). The actual construction of such a tour has been sketched earlier, and will be discussed in more detail in later sections.

The conversion from the postman's graph to the graph of shortest paths between nodes of odd degree may be accomplished by means of an efficient shortest path algorithm [Hu, 1967]. In a more recent paper, Edmonds and Johnson [1973] have presented a single algorithm which combines the shortest path conversion with the matching algorithm and permits solving the problem directly on the original graph.

2.5 Properties of an Optimal Tour

The contributions discussed thus far provide a complete solution to the problem of routing a single collection vehicle in a predetermined collection area with no one-way streets, in such a way that the traverse cost is minimized. This work will be further discussed in Chapter 3. Some additional work has been done which provides the fundamental approach to the solution of the same problem when some or all of the streets in the network are one-way. Discussion of this research will also be deferred to the appropriate sections of Chapter 3.

We close this chapter with a summary of some of the properties of an optimal tour on an undirected graph which have been discussed. These properties are of particular importance for the designer of vehicle routes who is not using the computer techniques discussed in later chapters. They permit the identification of potential improvements in any existing or proposed tour.

1. Any tour, whether optimal or not, can be represented by a graph of the street segments, with retracings (traverses without collection) indicated as duplicated edges.

2. The graph of any tour, whether optimal or not, will be unicursal.

3. In a unicursal graph, every node is of even degree.

4. In the graph of an optimal tour, the duplicated edges form paths which connect nodes which were originally of odd degree.

5. In the graph of an optimal tour, no edge is duplicated more than once.

6. In every cycle on the graph of an optimal tour, the sum of the lengths (traverse costs) of duplicated edges will be less than or equal to the sum of the lengths of unduplicated edges.

7. In the graph of an optimal tour, the path of duplicated edges connecting two nodes originally of odd degree must be the shortest possible path connecting these two nodes.

8. The graph of an optimal tour may be determined by constructing a modified graph containing only the nodes of odd degree from the original graph, with connecting edges representing shortest paths between them, and solving the matching problem on this new graph. The edges which appear in the optimal matching represent the shortest paths which must be duplicated in the original graph.

9. A tour on a unicursal graph may be constructed by first constructing a tour on any cycle of the graph. If this cycle does not travel over every edge of the graph, a second cycle of untraveled edges may be found which touches one node in the first cycle. The first cycle may then be broken at this node, and the second cycle inserted into the tour at this point. This procedure may be continued until every edge is traveled.

CHAPTER 3: SINGLE VEHICLE TOURS

The material presented in Chapter 2 provides the basis for the construction of optimal tours for single vehicles on a district consisting entirely of two-way streets. The first section of this chapter describes the techniques for this construction, and discusses variations of the problem which can also be treated with the same algorithms. Succeeding sections of this chapter develop and discuss algorithms for the construction of single vehicle tours in districts consisting entirely of one-way streets and in mixed districts containing both one-way and two-way streets.

3.1 Single Vehicle Tours on Undirected Graphs

We assume at the outset that the tour designer has pre-determined a district in which the tour is to be constructed. This district generates an amount of solid waste which is less than or equal to the capacity of the collection vehicle, so that collection can be performed in a single trip. The district may be determined by means of the multiple-tour algorithm described in Chapter 4, by other heuristics suggested in Chapter 4, or by any other means available. It may be, of course, that the designer is simply re-designing a tour which already exists, so that the district has been previously defined by practice.

It is then necessary to determine the traverse cost of every edge (street segment) in the district. As discussed earlier, this value should represent the total cost of travelling over the street segment without collecting. It includes the time cost associated with wages, and the distance cost associated with gasoline usage and vehicle depreciation. It is not necessary that the value used for traverse cost be a true cost; only that it be proportional to the true cost so that traverse costs of all the edges are correct relative to one another. Thus, if it is felt that there are only minor variations in vehicle speed over the various segments, then true cost will be proportional to segment length, and length may be used as the traverse cost. If the district includes major thoroughfares as well as small residential streets, so that traverse speeds are quite different, it may be possible to subdivide the streets into two or more categories, with different average speeds for each category, and to calculate traverse costs on the basis of distance using a different weighting factor for streets in each category.

With traverse costs determined for the district, the problem is ready for solution by computer. The input data for the computer

code represent the graph of the network. Nodes are identified by numbering them (in any order); edges are identified by stating the two nodes on which they are incident. The input data consist of one punched card for each edge, giving the two incident nodes and the traverse cost. Complete details of the input format are contained in Appendix A of this report.

It should be pointed out that the input data may be simplified by two omissions. Since every dead-end street must be retraced and the collection of such a street may be done anytime the tour passes its entrance, dead-end streets may be omitted from the computer data, and the designer may put them back into the tour in writing up the final solution. Similarly, it may readily be seen that any node of degree two is merely an intermediate point on a path made up of the two incident edges; since no retracing starts or ends at such a node (it is of even degree), it may be omitted and the two edges replaced by a single edge whose traverse cost is the sum of the costs of the replaced edges.

The computer code begins by identifying all nodes of odd degree in the district. It then applies a shortest path technique to develop a table indicating the shortest paths between all of these nodes. This information is in turn used for entry into a matching algorithm.[1] The result of the matching is the list of edges which must be traversed in order to convert the street network graph into a unicursal graph with minimum traverse cost.

This list of traversed edges is all that the designer needs in order to construct an optimal tour. There are, of course, many different tours which may be constructed on the resulting unicursal graph, each of which is optimal in the sense that traverse cost is minimized. The designer may have many additional objectives, principally relating to turns and order of collection, which will suggest preferences among the optimal tours. For most of these requirements, the designer's judgement is probably significantly better than any computer rules which can be devised using current knowledge. Thus, one of the outputs from the computer code is the list of traversed edges, provided so that the designer may use trial and error procedures to determine a satisfactory tour.

In addition, however, the computer code determines two of the possible optimal tours and provides them for the designer's information. An algorithm for constructing a tour on an undirected

---

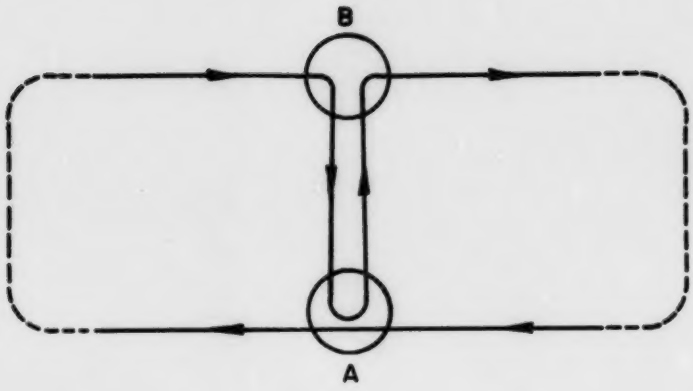
[1] The computer code used for matching is not that of Edmonds; rather a modified algorithm coded by Lee J. White [1967] is used because of its coding efficiency.

unicursal graph has already been described. For convenience of coding, the algorithm used in our program is a modification of a second approach suggested by Edmonds and Johnson [1973]. This technique will be described in the discussion of tours on graphs when some streets are directed. Our computer code uses the tour-generating algorithm to construct one euler tour on the unicursal graph which results by duplicating the edges found in the matching algorithm, and then prints out this tour.

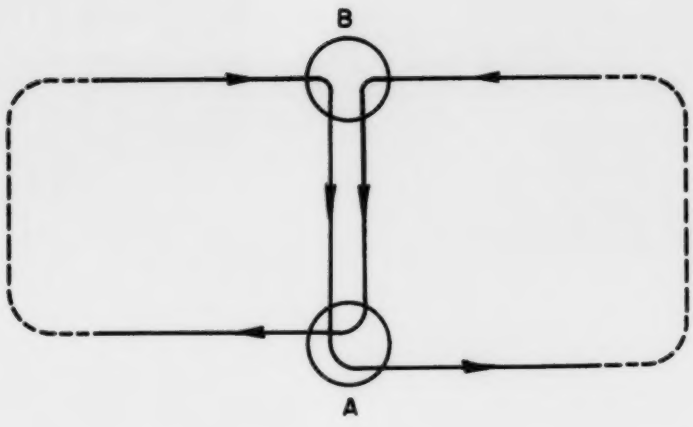
One of the characteristics of the tour found by construction in this manner is that it may contain a number of U-turns, which are generally undesirable. Except for those on dead-end streets, all U-turns in such a tour may be eliminated by a rather simple procedure, shown in Figure 9. Figure 9a indicates a portion of a tour which has a U-turn at node A. The dashed lines represent the remainder of the graph; since the tour must be a closed cycle, the dashed lines must close the loops as shown. Since we are assuming no dead-end streets, the degree of nodes A and B must be at least 4 (we also assume that the street segment AB is retraced). Figure 9b illustrates the elimination of the U-turn by reversing the direction of travel on one of the two loops and interconnecting them at node A. Such a procedure may be followed to eliminate every U-turn in any tour which has been constructed, except for U-turns on dead-end streets. Our computer code applies this procedure to the tour which it has generated, and prints out a second tour which is simply the first tour with U-turns eliminated, and with the same traverse cost.

It is important to emphasize the flexibility which this approach provides. The specification of edges to be traversed defines a large family of tours, each of which has minimum traverse cost. Two such tours are provided as samples, but the designer is, in fact, free to select any tour which he can construct on the resulting unicursal graph. He may use the cycle-building technique for constructing a tour, or he may use trial-and-error. He may begin with one tour, and by moving cycles from one part of the tour to another, he may modify the tour to better satisfy his objectives. This flexibility permits him to insure, for example, that major thoroughfares are not travelled during rush hours. It also permits him to observe restrictions on his solutions which are not able to be handled by the computer algorithms. For example, no provision is contained in the computer codes for intersections at which left-turns are prohibited. In most cases, the designer will find it fairly easy to construct a tour which observes such turn restrictions.

In rare cases, the designer will not be able to find a tour on the optimal unicursal graph which satisfies his additional constraints. In such a case, he must then consider increasing the traverse cost of the tour. This is easiest to do if the requirement that the



(a)



(b)

Figure 9: Elimination of a U-Turn

graph remain unicursal is kept in mind. This implies that, in every such change, duplicated edges must be added in complete cycles (in order to keep every node of even degree), or that duplicated and non-duplicated edges must be exchanged in a complete cycle.

A variation of the single-tour problem which occurs rather frequently involves street segments (edges) on which collection is not necessary. Such edges will only be traversed if they lie in the shortest path between two matched nodes of odd degree. Furthermore, the degree of a node is determined by the number of required entrances and exits from the node, and therefore does not include edges which need not be used. In effect, these edges are treated as potential retracings which, if they are used, will change the degree of a node in exactly the same manner as the duplication of a required edge and, if they are not used, will have no effect on the graph. The input form for the computer code includes an optional indicator for edges on which collection is not required. Edges so indicated are not used in calculating the degree of each node, but are used in determining shortest paths between nodes of odd degree. Thus, the matching algorithm determines optimally whether these edges should be traversed. Our algorithm may fail, however, if the omission of all optional edges results in a graph which is not connected.

There is one important sense in which the tour determined as described above may be non-optimal. Thus far, we have assumed that what is always desired is a tour which starts and finishes at the same node (intersection) in the collection district. Since the tour forms a closed cycle, it is immaterial which starting node is chosen; clearly, the most likely intersection is that which is closest to the depot. We have not considered, however, the possibility that the tour may start at one node and end at another node. There are two different circumstances in which this may be the optimal choice. Suppose that we choose to begin the tour at one node of odd degree and terminate it at another node of odd degree. These two nodes may then be left of odd degree within the collection district, because the entry from the depot and exit to the depot in fact convert them to even-degree nodes. Thus, we eliminate one path of duplicated edges; but, on the other hand, we add some length to the tour because one of the two selected nodes is further away from the depot. If the eliminated traverse cost is greater than the added travel to the depot, this "open tour" is an improvement.

The other case in which a closed cycle tour is likely to be non-optimal is the case in which the vehicle begins its tour at the start of the work-day from a garage, and terminates the tour at a depot which is not at the same location. Clearly, in this case, it may be desirable to begin the collection at a node near the garage, and to end it at a node near the depot.

Our algorithm may be used to solve this extended problem by means of some small modifications of the data.[2] The collection district is represented as before, but, in addition, we represent the depot by two nodes. One node may be termed the entrance to the depot, and the other the exit from the depot. These two nodes are joined by a single edge, which has a length equivalent to the traverse cost from the depot to the closest point in the collection district and back. In addition, edges are added between the depot entrance and any point in the collection district which is a permissible terminal point of the tour, and between the depot exit and any point in the collection district which is a permissible starting point of the tour. The lengths of these edges represent the actual traverse costs of the corresponding trips. These edges are indicated as optional edges, while the edge connecting the two nodes of the depot is a required edge. With this additional data, the matching algorithm will determine the optimal tour, including the depot.

To see why this is so, we must recognize that the two nodes representing the depot are of odd degree, because the only required edge touching them is the edge between them. These nodes must therefore be matched. They may be matched with each other, in which case the edge joining them is "retraced" at a cost equivalent to the trip to the closest point in the collection district and back; and, because the edges connecting the depot to the collection district are all optional, the depot is "disconnected" from the collection district and a tour is found in the collection district independent of the depot. On the other hand, the two nodes representing the depot may each be matched with some node in the collection district, in which case the added cost is the cost of traversing the two optional edges which are used. Thus, the matching algorithm will select the best arrangement to minimize traverse cost including the trip to the depot.

It has been pointed out throughout this report that the algorithms described above are valid only for networks with no one-way streets. The only reason for this is the fact that the matching algorithm may develop cycles which are impossible due to the direction of the one-way streets. Although an algorithm for a network with some one-way streets is discussed in a later section of this chapter, it is computationally much less efficient than the techniques for all two-way streets. Thus, if a district has only a few, widely separated one-way streets, it is probably worthwhile to use the techniques discussed above and attempt to

---

[2] We are indebted to Mr. Daniel Sheer, The Johns Hopkins University, for the suggestion which led to this representation.

find a tour which follows one-way streets in the correct direction. If this turns out to be impossible, then the less efficient approach may be used.

One additional important use of these techniques should be mentioned. Thus far, we have been assuming that the district consists of one truck-load. It is also possible to use the single vehicle algorithms for finding a route (full-day's schedule) for a truck, when the route consists of two tours. To do this, we simply consider the area to be collected on both tours to be a single district, and solve the problem as if it were a single tour. In constructing the tour, however, the designer is usually able to determine a sequence of edges such that some node nearest to the depot is visited about midway in the tour. The tour may be broken at this point for a trip to the depot, and then resumed again by the vehicle on returning from the depot. Theoretically, of course, this approach may be extended to 3-tour routes and beyond; however, it becomes increasingly difficult to construct tours which visit the appropriate nodes at the correct points of the tour.

3.2 Single Vehicle Tours on Directed Graphs

The problem of determining an optimal tour in a network consisting of only one-way streets seems, at first consideration, to be so rare as to be completely unimportant. There is, however, one situation in which this problem may be quite common in some cities. If it is the general policy in a city or in portions of a city to collect on each side of a street separately, then each two-way street is represented in the graph as two arcs (directed links) directed oppositely, and each one-way street is represented as two arcs directed the same way. This will result in a graph made up entirely of arcs, all of which must be traveled.

In the completely undirected case, the necessary and sufficient conditions for unicursality are that the graph is connected, and that every node is of even degree. It may also be shown that in the completely directed case, the necessary and sufficient conditions for unicursality are that the graph is connected, and that the number of arcs directed into each node is equal to the number of arcs directed out of that node. If the original street network does not satisfy these conditions, then some arcs must be retraced in order to attain unicursality. In the undirected case, the retraced edges formed shortest paths between nodes of odd degree. In the directed case, the retraced arcs form shortest paths (obeying the assigned directions of the arcs), leading from nodes with an excess of incoming arcs to nodes with an excess of outgoing arcs.

We have seen that the even-degree property required of all nodes

for unicursality in the undirected case is equivalent to a requirement that the entire graph be made up of cycles, with each edge or retracing being contained in exactly one cycle. The solution technique, which finds paths connecting odd-degree nodes, in effect creates these cycles. In the directed case, the incoming-outgoing balance requirement is also equivalent to a requirement that the entire graph be made up of directed cycles (which, in graph-theory jargon, are called circuits) with each arc or retracing being contained in exactly one circuit. In this case, however, it is easier to construct the solution by direct construction of the circuits.

A well-known algorithm which we may use for this problem is the "out-of-kilter" algorithm. Consider a directed network in which the arcs represent pipes. Associated with each pipe are three numbers: a unit cost of flow, an upper bound on flow (capacity), and a lower bound on flow. The lower bound may be zero, and the upper bound may be infinity. Given these quantities, the out-of-kilter algorithm finds flows in all pipes, such that the upper and lower bounds on flow are satisfied, inflow equals outflow at each node,[3] and the total cost of the flow is minimized. Clearly if the lower bounds on all pipes are zero, then the optimal solution has zero flow. However, if there are any positive lower bounds on flows, these bounds provide a "driving force" which may force flows in other pipes. In order to maintain the condition that inflow equals outflow at every node, the resulting flows must be in the form of closed circuits. Since the algorithm minimizes total cost, the circuits are minimum cost circuits which satisfy the requirements.

To solve our problem of finding a minimum cost unicursal graph, we represent the original directed street network as a directed graph. The unit flow cost of each arc is its traverse cost, and the lower bound on flow in each arc is unity. All upper bounds are infinity. The out-of-kilter algorithm will then find circuits which satisfy continuity of flow; this will result in flows greater than one on arcs where such flows are necessary in order to make inflow equal to outflow. Each time the flow in an arc is increased by one, the total cost is increased by the traverse cost of the arc. We interpret the flows in an arc as the number of times the arc must be traveled; flows greater than one imply retracings of the arc.

Thus, the out-of-kilter algorithm serves the identical function in the directed case as the matching algorithm in the undirected

---

[3] More precisely, the out-of-kilter algorithm maintains the difference between inflow and outflow at a node which is provided by the input initial flows. This permits nodes to be sources or sinks of flow. In our application, all initial flows are zero, so inflow always equals outflow.

case: it determines the required arc retracings such that the resulting graph is unicursal and the traverse cost is minimized. It is important to note that one property of the optimal unicursal graph in the undirected case does not extend into the directed case: arcs in the directed case may be retraced more than once. This is true because the difference between the number of incoming and outgoing arcs may be greater than one, and the least costly way of achieving the required balance may be to travel over the same directed path several times.

It should also be pointed out that the extensions discussed in the undirected case may also be treated in the directed case. Arcs which have no collection requirement, but which may be traversed if needed to achieve balance, are represented in the out-of-kilter graph with lower bounds of zero instead of one. There is also a way of including the depot in the tour.

The required expansion of the graph to include the depot is shown in Figure 10, where the numbers beside the arcs are lower bounds. We add to the graph of the collection district two additional nodes representing the depot. One of these nodes is the "entrance" to the depot, and the other is the "exit" from the depot. We add a directed arc from the entrance node to the exit node, with a lower bound of one. The cost to be assigned to this arc is discussed later. In addition, we add directed arcs from the exit of the depot to any node in the collection district which we wish to consider as a possible starting node for the tour. These arcs have a lower bound of zero (since all of them need not be travelled), and a cost equal to their traverse cost. Similarly, we add directed arcs from any node in the district which we wish to consider as a possible finishing node to the entrance of the depot. These arcs also have a lower bound of zero and a cost equal to their traverse cost. We then enter the out-of-kilter algorithm with this enlarged graph.

The lower bound of one on the arc from the depot entrance to the depot exit forces the inclusion of the depot in a circuit. Thus, at least one arc from the collection district to the depot entrance, and at least one arc from the depot exit to the collection district will be used. These arcs, in turn, will force modification of the flows within the collection district in order to maintain inflow equal to outflow. Again, since the out-of-kilter algorithm seeks minimum cost solutions, additional flow will occur only where it is necessary to maintain the required conditions, at minimum traverse cost.

The cost of the arc from depot entrance to depot exit may be set at zero. However, particularly in collection districts which are close to the depot, this may result in a flow greater than one through the depot, which implies more than one visit to the depot during the tour. This can occur only when it is, in fact, less

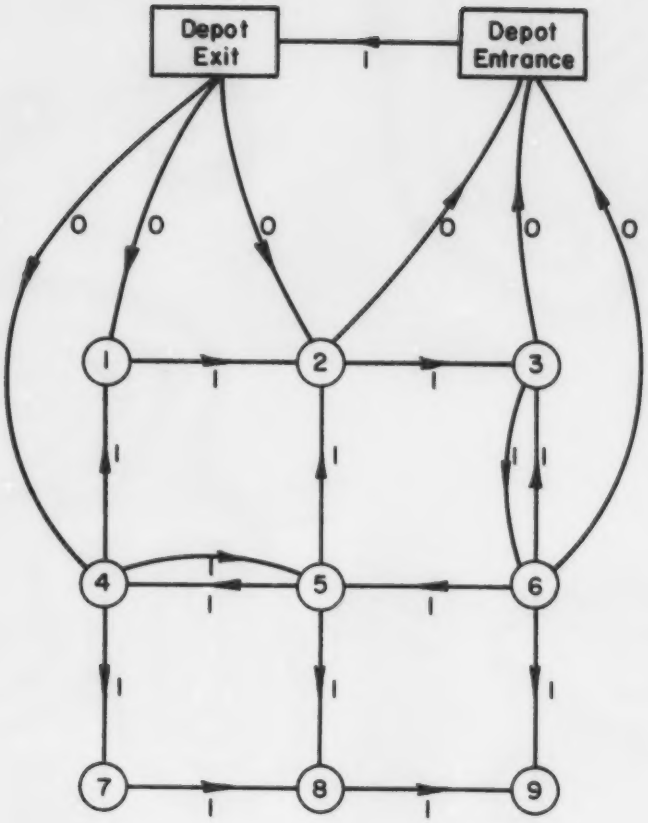


Figure 10: Representation of a Directed Case with Depot (Numbers adjacent to arcs are lower bounds; all upper bounds are infinite. Costs, which are not shown, are traverse costs).

costly to make additional visits to the depot as a way to get from one portion of the collection district to another, rather than making some lengthy traverse within the district. In order to provide a more realistic solution in such cases, it may be desirable to assign a cost to the arc from depot entrance to depot exit which represents the actual cost of visiting the depot (incinerator, landfill, transfer station, etc.). With such costs assigned, if the solution still provides a flow greater than one through the depot, it simply indicates that additional visits to the depot remain less costly, even considering the added expense. We may, of course, insure that there is only one visit to the depot by assigning a particularly large cost to the arc from entrance to exit.

In any case, the result of applying the out-of-kilter algorithm will be a flow in each arc, representing the total number of times (original trip plus retracings) that the arc is to be travelled. One or more visits to the depot will be included. The graph which may then be drawn with the indicated number of retracings of each arc is unicursal, and has minimum cost (including the cost of passing through the depot).

If we seek true optimality, arcs to and from the depot should be provided at every node in the collection district, so that the algorithm may select from among all possible alternatives. This has not been shown in Figure 10, because it makes the graph very difficult to understand. In most real cases there are only a few likely candidates for starting and finishing nodes, which can be determined easily by inspection.

It is now possible to make a slightly surprising statement about one of the cases with which we began this section. If we have a district composed entirely of two-way streets, and it is our policy to collect on opposite sides of the street separately, we have pointed out that we may represent each street by two oppositely directed arcs, both of which must be travelled. However, in this case, we need no algorithm at all, for the graph is already unicursal, and it is possible to construct a tour which has no traverses at all! At each intersection, every street is represented by one incoming and one outgoing arc; thus the number of incoming arcs is always equal to the number of outgoing arcs. Assuming that the graph is connected, the unicursality conditions are satisfied. Furthermore, the start and finish of the tour must be at the same node, since no distance in the collection district can be eliminated by starting and finishing at different nodes. Obviously, the proper node to select for start and finish is the one closest to the depot.

The need for the directed case algorithm occurs in the following two situations:

a). when all streets are, in fact, one-way.

b). when there are some one-way streets, and all two-way streets must be collected separately on each side, thus creating nodes where the number of incoming arcs is not equal to the number of outgoing arcs.

Our discussion thus far in this section has been concerned with the construction of an optimal unicursal graph. We now turn to the construction of an euler tour on the resulting graph. A unicursal graph in the directed case is made up of circuits, just as it consisted of cycles in the undirected case. Thus, it it possible to construct an euler tour by beginning with a circuit and breaking it to insert an additional circuit. However, there is a different approach which is somewhat easier in this case, and which extends to the mixed case of the next section much more simply. This algorithm, also due to Edmonds and Johnson [1973], is described below.

We begin the algorithm by constructing a spanning arborescence whose root is at the node where the tour starts and finishes. A spanning arborescence is a connected set of arcs such that every node of the graph is touched by at least one of the arcs, and there is exactly one of the arcs directed outward from every node except the root. A spanning arborescence has no circuits, and has a path from every node in the graph to the root. There are often many possible spanning arborescences in a graph; any one of them will serve our purpose. It is fairly easy to construct a spanning arborescence. We begin with the root node, which we place in the arborescence. We then add to the arborescence any arc directed towards a node in the arborescence and away from a node not in the arborescence, and we also include in the arborescence the node at the far end of the arc. We simply continue in this fashion, adding arcs which are directed towards the arborescence and their end points until every node is in the arborescence. An example of a spanning arborescence on a unicursal graph is shown in Figure 11.

The important property of the arborescence is that it provides a path from every node in the graph back to the root. If we simply tried to build an euler tour by starting at the root and travelling over arcs randomly, the difficulty which we would be likely to encounter is that we would use up all paths to the root from a portion of the graph, without having travelled over all arcs in that portion. The arborescence may be used, however, to reserve a path to the root until it is permissible to use it. Once the arborescence has been constructed, we may trace an euler tour by the following procedure. We begin at the root and leave by any arc directed outward. When we arrive at the next node, we may leave by any outward-directed arc except the one which is

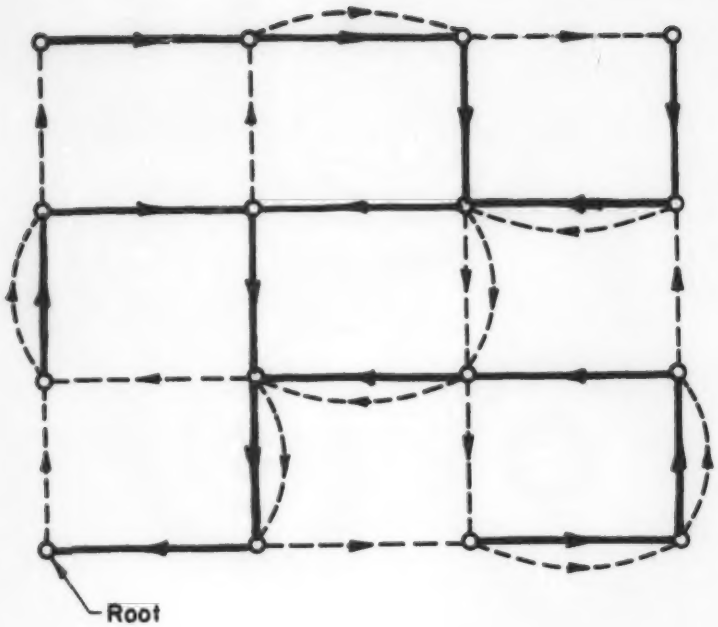


Figure 11: A Spanning Arborescence on a Unicursal Graph

in the arborescence, unless there is no other outward-directed arc from that node. In other words, at any node, outgoing arcs which are not in the arborescence take precedence over the one outgoing arc which is in the arborescence. At each node we may select any outgoing arc we wish with the above exception; thus we may exercise judgement and preferences in constructing the tour.

The above procedure works because the number of incoming arcs at every node is equal to the number of outgoing arcs. Thus, when we leave a node by an arc in the arborescence, it must have been true that we entered the node by the last available incoming arc, and we can never again reach that node. We have, in effect, preserved the path to the root until last.

A computer code has been constructed for the determination of optimal tours in directed graphs. The input format for this program is similar to that for the undirected case: one card is provided for each arc, listing the two nodes connected by the arc and the traverse cost. In this case, however, the order in which the two nodes are given implies the direction in which the arc may be travelled. There is also an option to indicate arcs which have no collection and therefore need not be travelled.

The computer code begins by setting up the problem in the form of the out-of-kilter algorithm. Arcs which must be travelled are given lower bounds of one, while those which need not be travelled are given lower bounds of zero. All upper bounds are set to infinity, and traverse costs are provided as flow costs. The out-of-kilter algorithm is then used to solve the problem and provide a unicursal graph. The result is printed out, so that the designer may construct his own tours, if he so desires. In addition, the spanning arborescence algorithm is used to construct a single euler tour, which is printed as a sample for the designer.

There is no U-turn elimination step in the computer code for the directed case. This is because the elimination of U-turns usually involves reversing the direction of travel in some cycle, which is not possible because the cycles are directed. The designer may attempt to minimize the number of U-turns, but there is no guarantee that the optimal unicursal graph will have an euler tour with no U-turns. In order to completely eliminate U-turns it may be necessary to add additional traverse cost.

3.3 Single Vehicle Tours in Mixed Graphs

The mixed graph case, when some links are directed and others are undirected, is the most difficult of the three cases to solve.

This is principally due to the complexity of the necessary and sufficient conditions for unicursality, which have been given by Ford and Fulkerson [1962]. They are:

a) The graph must be connected.

b) The total number of links incident on each node of the graph must be even.

c) For every node, and every subset of nodes, the number of undirected links incident on the node (or subset) must be greater than or equal to the difference in number between incoming and outgoing directed links.

The first two of these conditions are familiar requirements; the third is somewhat different. We consider this condition as it applies to a single node first. Figure 12a shows a node which has 2 incoming arcs, 1 outgoing arc, and 1 undirected edge. The node is of even degree, and the number of undirected links is equal to the difference between the number of incoming and outgoing links. Thus, the node satisfies all our requirements. We note that by using the undirected link as an exit from the node, we can satisfy the obvious requirement that number of entrances equals number of exits. In Figure 12b, the indicated node has 3 incoming arcs, no outgoing arcs, and 1 undirected edge. Although the node is of even degree, it does not satisfy condition c), since the difference between the number of incoming and outgoing arcs is three, while the number of undirected edges is only one. Note that even though we use the undirected edge as an exit, we cannot provide enough exits from the node.

Condition c) must hold, not only for every node, but also for every subset of nodes. The need for this additional requirement is shown in Figure 13. The graph in this figure is connected, every node is of even degree, and condition c) is also satisfied at every node. However, if we consider the region of the graph defined by nodes 1, 2, and 5, we note that we will be required to enter this region three times (along the three directed arcs), and we have only one undirected edge available along which to exit from the region. Thus, it is clearly impossible to trace an euler tour in this graph. In effect, condition c) says that we must be able to treat any region of the graph as a sort of "super-node", and it, too, must satisfy the requirement for an equal number of entrances and exits.

An algorithm for duplicating links in a non-unicursal mixed graph such that the resulting graph is unicursal with a minimum traverse cost was developed as part of this project. The mathematical details of this complex algorithm have been given in Appendix B of the first annual report, and will not be repeated

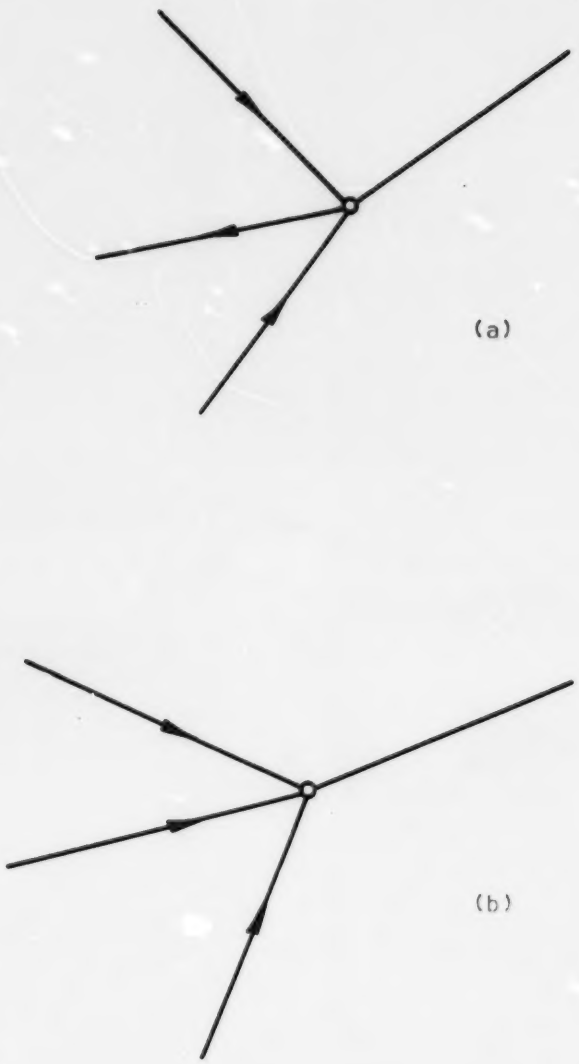


Figure 12: Unicursality Conditions on Nodes in a Mixed Graph

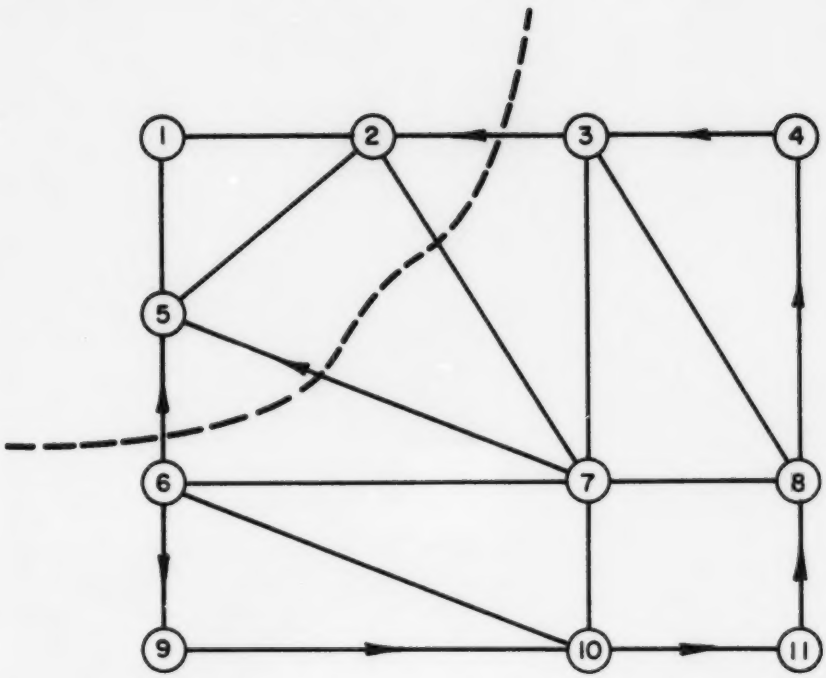


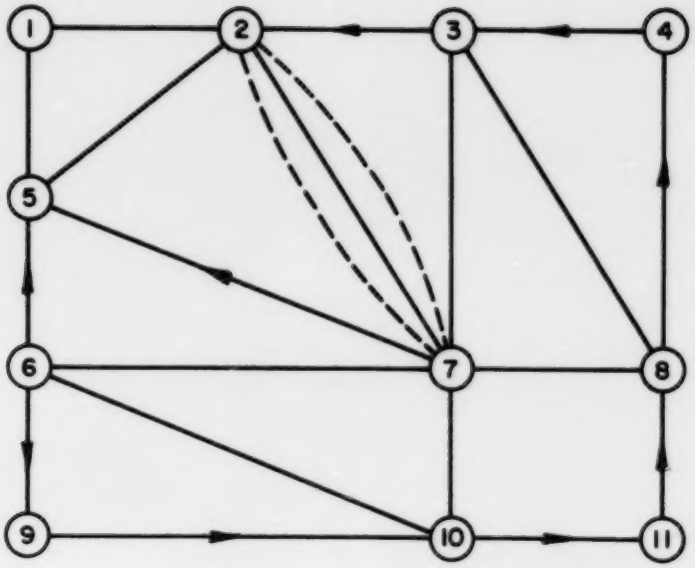
Figure 13: A Mixed Graph Which is not Unicursal

here. We simply point out that this technique specifies the links which must be retraced in order to satisfy the mixed-graph unicursality conditions. As in the completely directed case, links may be duplicated more than once. Unlike either of the two previous cases, duplicated links need not form paths between odd-degree or unbalanced nodes. For example, in Figure 13, the graph may be made unicursal by two duplications of the edge between nodes 2 and 7, as shown in Figure 14a.

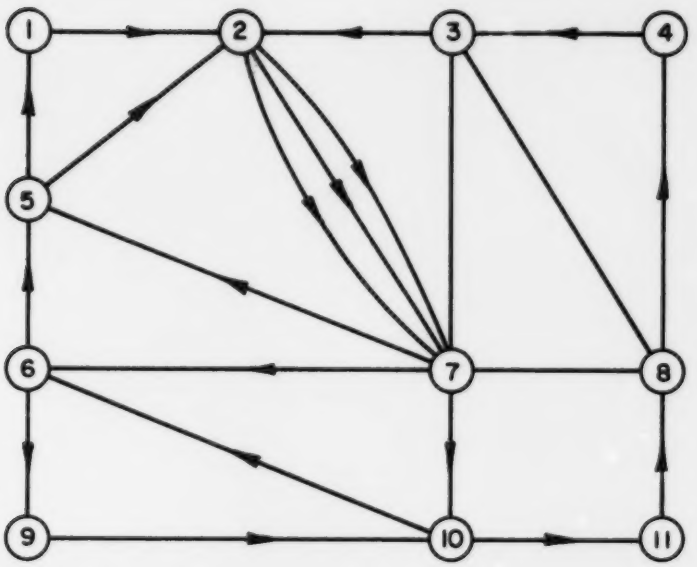
Once a minimum traverse cost unicursal graph has been obtained, the technique by which a tour may be constructed is also somewhat more complicated. It involves three steps, the last two being quite similar to the technique used in the completely directed case. Before we may build a spanning arborescence, however, we must make one modification on the graph. The arborescence technique requires that, for every node which is touched by any directed arc, the number of arcs directed into the node be equal to the number of arcs directed out of the node. The solution of the unicursality algorithm will not provide this condition, called symmetry. However, it is a property of a unicursal graph that symmetry may always be created by assigning appropriate direction to some of the undirected edges. This may be done by trial and error, recognizing that the symmetry requirement is equivalent to requiring that every arc be contained in a circuit of arcs. It may also be done by using the out-of-kilter algorithm. We construct a new graph by representing every undirected link in our unicursal graph by two oppositely-directed arcs, and retaining all of the directed links unchanged. On all of the originally directed links we place a lower bound of one, and an upper bound of one. On all of the arcs representing originally undirected links we place a lower bound of zero and an upper of one. The unit costs on all links are their traverse costs. Solving the out-of-kilter algorithm on this graph will force a unit flow through each originally directed arc. Since the algorithm retains flow balance at every node, it will induce circuits, using some arcs representing originally undirected links. These links are then assigned direction in the unicursal graph, which results in symmetry. Figure 14b shows the result of this assignment. We note that some links remain undirected, and that these form closed cycles. At every node, the number of arcs directed inward is equal to the number directed outward, which is the symmetry requirement.

It is important to note that making the graph symmetric has not added any further duplicate links. The graph remains unicursal, and thus there is still an euler tour with minimum traverse cost. It is true, however, that there may be more than one way of making the graph symmetric, and each symmetric solution will ultimately exclude some euler tours which would be possible on a different symmetric solution.

Once the graph is both unicursal and symmetric, we may proceed in



(a)



(b)

Figure 14: The Graph of Figure 13 made Unicursal (a) and Symmetric (b).

much the same fashion as with the totally directed case. We first build a spanning arborescence; then construct a tour using arcs in the arborescence last. However, the rule for building a spanning arborescence is somewhat modified [Edmonds and Johnson, 1973].

We begin to build the arborescence in exactly the same fashion as before. Starting at the root, we add arcs directed toward some node in the arborescence, away from some node not in the arborescence. We continue in this fashion until we can go no further, using only appropriately directed arcs. It may be that we now have a spanning arborescence, in which case we may go on to construct a tour. However, it is likely that the arborescence we have constructed is not spanning (that is, some nodes are not yet included). This will occur if there are two or more sections of the graph containing directed arcs, separated by a region containing only undirected edges. We now identify any complete cycle of undirected edges which touches some node in the arborescence. The existence of such a cycle is guaranteed by the unicursality and symmetry of the graph. We assign direction to every edge in this cycle, in consistent fashion, so that we now have a circuit. These edges are now treated as directed arcs, and will permit us to extend the arborescence further. We continue assigning direction to cycles as needed until we have completed the spanning arborescence.

The above algorithm constructs a spanning arborescence by using directed arcs wherever possible. Whenever an undirected edge must be used, direction is assigned to an entire cycle of edges, in order to insure that the unicursality conditions continue to be satisfied.

With the spanning arborescence completed and, perhaps, some additional edges converted to arcs by the assignment of direction, we may now construct an euler tour. We begin at the root, and leave using any undirected link, if there is one. If there is not an undirected link, we use an outward-directed link. When we arrive at the next node, we leave by an undirected link if there is one. Otherwise, we leave by any outward-directed link, except the one in the arborescence, unless that is the only remaining one. In other words, the priority at each node is in order: undirected link, outward-directed link not in the arborescence, outward-directed link in the arborescence. This approach will always result in the construction of an euler tour.

Our computer code for the mixed graph case uses an input format similar to that used in both of the previous cases. One card is provided for each link, indicating the two end points and the traverse cost. An additional indicator is used to designate directed links. The code first applies the unicursality algorithm, and prints the list of links which must be retraced. It then uses the out-of-kilter algorithm to attain symmetry, and prints

a list of edges which have become directed arcs. Finally, it constructs a spanning arborescence, determines an euler tour, and prints out the tour.

The unicursality algorithm used in this code utilizes a complex linear programming model with column generation. Solutions of this model are meaningful only when the variables take on integer values. No proof has yet been developed that this is always the case, although no real counterexamples have been found. It is thus possible that the algorithm will fail because of noninteger values for the number of retracings of some street segments.

The current computer code has one additional restriction, not contained in the two other cases. There is no provision in the mixed-graph code for street segments which do not require service; it is assumed that collection must take place on every street in the network. The modifications required in the algorithm to permit optional streets are discussed in Section 4.4 of Appendix B of the first annual report. These modifications have not been coded at present.

It has been pointed out before that the mixed-case code is much less efficient than the undirected case code. If there is only a single one-way street in the district, the undirected algorithm should be used. This is always possible, since the tour determined in the undirected algorithm can be travelled in either direction, so the one-way street restriction can always be observed. Even when there are several one-way streets in the district, it is likely that a tour can be found using the undirected algorithm.

CHAPTER 4: MULTIPLE VEHICLE ROUTES

The algorithms of Chapter 3 are intended for determining a single tour of a vehicle in a district which has been pre-selected to consist of no more than a full truckload of solid waste. They may also be used to provide good (but not necessarily optimal) routes for a single vehicle when the routes consist of two tours. A much more general problem is the determination of routes for a number of vehicles simultaneously, on a larger area of the city. This chapter suggests a heuristic approach to this problem which may be performed with the aid of the previous algorithms, and describes a computer code which has been constructed to find good routes.

In constructing multiple vehicle routes, a number of problems are encountered in addition to the need to minimize the cost of extra traverses. The workload on the vehicles must be equalized, so that overtime is at a minimum and an equitable distribution of labor is achieved. Workload, however, is a function of both the amount of waste collected and the amount of travel done by the vehicle (which requires time). There are normally additional requirements on the arrangement of the individual districts. Generally, it is desirable that each district be made up of contiguous streets, and be a compact geographical region.

A rather simple heuristic rule for dividing an area into routes is based on the fact that a lower bound on the total traverse cost is provided by the unicursality algorithms of the previous chapter. Within the collection area (exclusive of trips to and from the depot), no set of trucks can perform collection in less total distance than that which would be travelled by a single truck if it were capable of collecting the entire area. Therefore, if we treat the entire area as a single district and determine the minimum traverse cost unicursal graph on the area, this traverse cost represents a lower bound on the cost for collection by several vehicles.

As has been suggested by Stricker [1970], a unicursal graph may be subdivided rather easily in such a way that each subgraph remains unicursal without additional duplication of edges. This suggests the following heuristic algorithm for determining multiple tours:

1. Treat the overall area as a single district, and find the minimum cost unicursal graph.

2. Subdivide the resulting graph into the appropriate number of districts, retaining unicursality of each district.

3. If the load to be collected in any resulting district is too great, modify the subdivision boundaries to reduce the load.

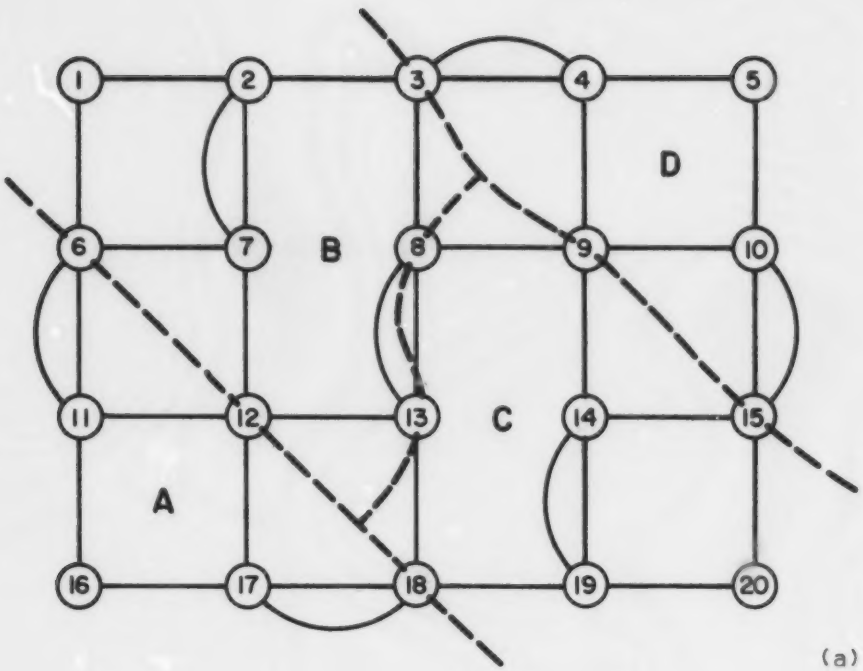
If we are able to perform the above steps, the resulting districts will also have minimum traverse cost. However, we have neglected the distances to and from the depot. Depending on how the districts are arranged, the traverse costs of these trips may be greater or less. In effect, the above heuristic determines optimal tours if we could drop the collection vehicles into their districts by helicopter and pick them up in the same fashion.

Figure 15 shows the subdivision of a unicursal graph into districts. The original network has been made unicursal by the use of the undirected algorithm of Chapter 3. The dashed lines represent boundaries of four districts which have then been constructed. In the process of drawing the district boundaries, we need only recognize that we must keep each node of even degree on both sides of the boundary; thus, we draw the boundaries through the intersections in such a way that an even number of streets touches each intersection on each side of the boundary.

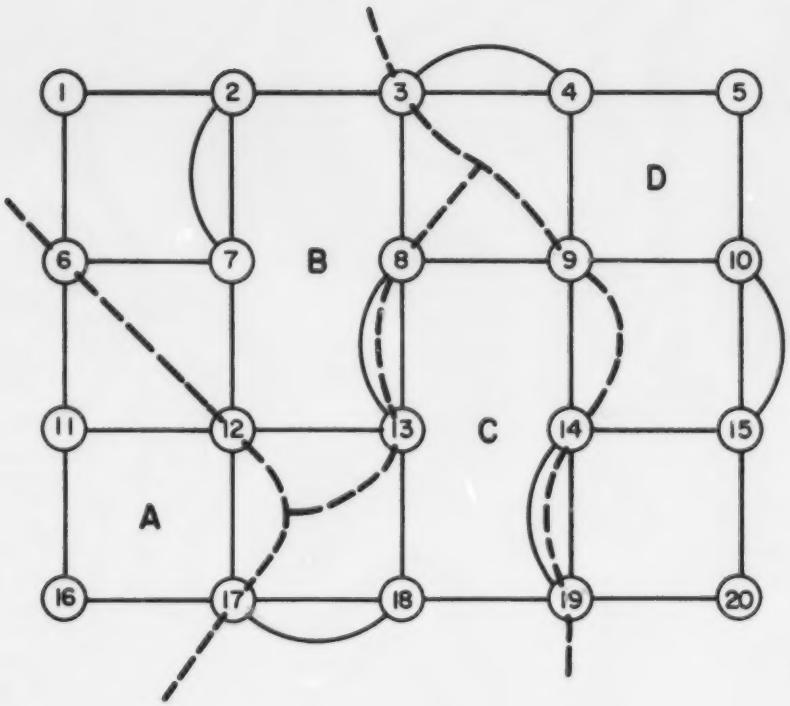
If inspection of the resulting districts indicates that some have excess load and others have too little load, we may modify the boundaries by shifting cycles of the graph from one district to another. In Figure 15b, district D has been made larger by shifting the cycle 14-15-20-19-14 out of district C, and district A has been made slightly smaller by shifting the cycle 17-18-17 into district C. We may continue to make modifications in this fashion until we are satisfied that the loads are balanced.

Since we constructed the districts so that they are each unicursal, an euler tour is possible in each without additional duplication of edges. Thus, with respect to travel within the districts, we have retained optimality. However, suppose the depot were located at (or near) node 16. Then the vehicle assigned to collect in district D would have to travel over some edges of the graph (perhaps 16-17-18-19) in order to reach his district; these retracings are in effect added duplications of those edges which we have not considered. We may, of course, keep the location of the depot in mind as we construct the districts, and try to minimize this additional cost using our own judgement.

The inability of the heuristic technique described above to directly consider the distances to and from the depot is its major drawback. A principal goal of this research has been to develop a better method, which would permit the depot location to influence the district boundaries directly, in such a way that travel to and from the depot is minimized. A heuristic algorithm for this purpose is discussed briefly in the following paragraphs



(a)



(b)

Figure 15: Districting on a Unicursal Graph

and in detail in Appendix B of this report.

The technique begins by constructing an optimal unicursal graph as described previously. This graph consists of a set of cycles, as do all unicursal graphs. One way of viewing the districting problem is that it consists of allocating the cycles among the districts. If districting is performed in this fashion, the unicursality of the individual districts is guaranteed; this, in fact, is the effect of the boundary construction of the procedure described above.

There is, of course, more than one way to define a set of cycles such that every edge of the graph is contained in exactly one cycle. Selecting a particular set of cycles restricts consideration to some subset of all the possible districts which could be defined. In order to provide the greatest possible flexibility, our algorithm works with the simplest possible set of cycles. The selected set of cycles forms a checkerboard pattern on the graph, as shown in Figure 1 of Appendix B. We restrict our consideration to tours which may be constructed by putting together these cycles, along with the paths to and from the depot.

Using the cycles defined in this fashion, we now construct a new graph. Every node in this graph represents one of the cycles in the original unicursal graph. Associated with each node is a load, which is the amount of solid waste to be collected within the represented cycle. Edges in this new graph represent the travel from one cycle to another. We include edges only for cycles which are adjacent; that is, cycles which have a node in common. Any two cycles of the original graph which have a node in common will be represented in the new graph by two nodes, with an edge connecting them. The length of the edge is zero, since there is no travel distance required between the two cycles. Thus, the graph which we have constructed so far has only zero-length edges.

We could, of course, extend the graph to show edges for two non-adjacent cycles. These edges would have a length representing the shortest distance between the two cycles. However, since it is generally desirable to construct tours which are compact, we have chosen to omit all paths between non-adjacent cycles.

We now append to this new graph a node representing the depot, with edges leading to every other node. These edges represent the distance which must be travelled if a vehicle goes from the depot to the closest point on the indicated cycle. The lengths of these edges are traverse costs. A single tour on the original graph may now be defined on this new graph. It consists of one edge from the depot to some node, and a collection of other nodes which are joined by edges. That is, the vehicle travels from the

depot to some cycle, collects on that cycle and a set of adjacent cycles, and returns to the depot along the same edge. Since the district consists of a set of adjacent cycles, such a tour is always possible.

The complete multi-truck problem may now be defined on this new graph. Suppose we are working with an area consisting of four truckloads. On the derived graph, we wish to find a spanning tree[1] which contains exactly four edges (which we will call arms) from the depot to the rest of the tree, such that the sum of the loads on the nodes adjoined to any one arm is less than the capacity of a vehicle, and the sum of the lengths of the arms is minimized. The tree must be spanning since nodes represent cycles and every cycle must be collected; the sum of the lengths of the arms must be minimized because that sum represents the additional traverse cost we incur beyond that which was required to make the original graph unicursal.

Our algorithm constructs this spanning tree by a systematic trial-and-error procedure. It first selects the four shortest possible arms, and attempts to construct the remainder of the tree. If no tree meeting the requirements can be found, the next shortest set of four arms is selected and the procedure repeats. This process is continued until a feasible solution is found. The algorithm is described in detail in Appendix B of this report.

The multiple-vehicle algorithm does not guarantee optimality, for several reasons. First, it restricts attention to one particular configuration of cycles, which may not, in fact, be optimal. Second, it considers only tours which start and finish at the same node. Third, the search for feasible districts with a particular set of arms is not exhaustive, and the algorithm may thus move on to a more costly set of arms when, in fact, a previous set had a feasible solution. However, our experience with the computer code indicates that it does, in fact, find quite good tours; in the majority of cases we have explored the districts found by the algorithm were at least as good as the best ones we could find by any other means.

The multiple-vehicle algorithm is valid only for undirected graphs. The general approach may be used manually on directed or mixed graphs, but we have found that the selection of the set of cycles is not easily systematized in these cases. There is one additional restriction for this algorithm: it cannot handle cases in which one street crosses another over a bridge without connection.[2]

---

[1] A tree is an arborescence on an undirected graph.

[2] In graph-theoretic terms, the algorithm can only handle planar graphs.

We have found that in most cases, satisfactory solutions for cases which violate the restrictions given above can be found by the following procedure. First, construct the optimal unicursal graph on the entire area, using the appropriate single-vehicle algorithm. Then solve for the districts using the multiple-vehicle algorithm, ignoring directions and treating bridges as intersections. Finally, apply the appropriate single-vehicle algorithm to each district to find the optimal tour. We emphasize that this procedure is far from optimal, and the multiple-vehicle algorithm alone should be used whenever possible.

The computer code for the multiple-vehicle algorithm is currently limited to graphs consisting of 90 nodes and 270 edges. In an average residential area, this amounts to approximately six truckloads, though there is wide variation in this number. We have found that natural boundaries (expressways, rivers, parks, etc.) occur with sufficient frequency in most areas to permit the designer to select areas sufficiently small to fit the computer code. The code may be extended for larger cases providing computer memory size is sufficient.

CHAPTER 5: SUMMARY AND DISCUSSION

The preceding chapters have described a set of techniques which may be used for determining districts and routes for collection vehicles. Many of the principles may be applied without the use of computers, by engineers or systems managers who perform the routing task manually. Computer codes which route individual trucks and which find good tours for multiple trucks have been constructed as part of this project, and their use is described in Appendix A.

Of extreme importance in the use of the methods developed herein is recognition of the fact that, used blindly, they will not provide perfect answers. In fact, their intent is quite different: they have been designed to aid a planner by providing information which will make his subjective decision-making somewhat easier. Thus, it should not be expected that one can completely automate the routing of collection vehicles using these techniques.

This philosophy was one of the major factors in the choice of routing methods based on the Chinese Postman's problem rather than on other approaches. Solutions using our algorithms include a listing of the particular street segments which must be traversed without collection in order to minimize distance or traverse cost, leaving a great deal of flexibility in the choice of particular tours in the hands of the designer. This permits the incorporation of intuition, detailed knowledge of the collection area, non-quantifiable requirements, and constraints which must be applied on an ad-hoc basis into the final routing design.

The computer codes have been applied to a small number of actual cases, using collection areas in the cities of Baltimore and Indianapolis. Computation times for the single-vehicle algorithms have usually been considerably less than one minute (on an IBM 360 computer), while the multiple-vehicle heuristic usually requires two to three minutes. The multiple-vehicle algorithm is limited to smaller cases than we had hoped; on the other hand, we found this restriction to be less damaging than we had expected, because natural boundaries occur so frequently in most cities.

Continued research, we believe, is necessary in several areas in order to fully realize the potential of computer-aided routing. In the single-vehicle routing models, the most important shortcoming is the inability to deal explicitly with intersection restrictions and costs. For example, an algorithm which will generate tours without violating 'no-left-turn' rules would be quite helpful. Since we believe that it is usually possible, with

little effort, to construct such tours on a unicursal graph by hand, such an algorithm is not of crucial importance, however. A more important extension, because the designer's intuition is not particularly good in this area, is the ability to weigh trade-offs between intersection costs and additional traverse costs. Coupled with an appropriate algorithm, this would permit the selection of an alternate (non-minimal) set of edges to be retraced if the added tour length incurred is less than the cost of left turns eliminated. In the current models, there is no simple way for the designer to find the second-best matching, nor to find any matching with a minimum number of left turns.

In the multiple-vehicle algorithm, there are probably many improvements which can be made in the heuristic rules incorporated into our code. Improvement is desirable in all three aspects of the algorithm: the "goodness" of the final result, the computation time required, and the computer storage required. Some (marginal) reduction in storage and computation time can be attained through the re-writing of the code. We suspect that significant storage reduction is unattainable, however, because minimal information to represent a large network requires a large amount of storage in any case. Improvement of the heuristic rules is probably the most promising area of research in multiple-vehicle routing.

On a larger scale, it might be desirable to have complete computer data management system coupled with the routing algorithms. Such a system would permit storage of an entire municipal street network, and would provide manipulation algorithms for selecting a particular region in the network and executing the desired routing algorithms on that region. A designer could thus try several different approaches to the routing problem, using the most appropriate algorithm in each case. It is not clear that the benefits from such a scheme would exceed its cost if it were devoted only to solid waste collection; however, when we consider the many other routing problems within the purview of a municipal government it becomes much more likely that the system would be worthwhile. Daily (or hourly) updating of the data base to reflect streets under repair, traffic snarls, etc., would permit use of the system to route emergency vehicles as well as for routine routing.

REFERENCES

Edmonds, Jack, 1965a. "The Chinese Postman's Problem," Bulletin of ORSA 13, Supplement 1.

Edmonds, Jack, 1965b. "Maximum Matching and a Polyhedron with 0,1 Vertices," Jour. of Research, National Bureau of Standards, 69B, January-June.

Edmonds, Jack, and Ellis L. Johnson. 1973. "Matching, Euler Tours, and the Chinese Postman," to appear, Mathematical Programming.

Euler, Leonhard. 1736. "Solutio Problematis ad Geometriam Situa Pertinentis," Comment. Academiae Sci. 1. Petropolitanae 8, pp 128-140. See also "The Konigsberg Bridges," Scientific American 189, 1953, pp 66-70.

Ford, L. R. Jr., and D. R. Fulkerson. 1962. Flows in Networks, Princeton University Press.

Glover, Fred. 1967. "Finding an Optimal Edge-Covering Tour of a Connected Graph (The Chinese Postman's Problem)," ORC 67-13, Operations Research Center, University of California, Berkeley.

Harary, Frank. 1969. Graph Theory, Addison Wesley.

Hu, T. C. 1967. "Revised Matrix Algorithms for Shortest Paths in a Network," J. SIAM 15, pp 207-218.

Kwan Mei-Ko. 1962. "Graphic Programming Using Odd or Even Points," Chinese Math. 1, pp 273-277.

Stricker, R. 1970. Public Sector Vehicle Routing: The Chinese Postman's Problem, Ph.D. Dissertation, Massachusetts Institute of Technology.

White, Lee J. 1967. A Parametric Study of Matchings and Coverings in Weighted Graphs, Ph.D. Dissertation, The University of Michigan.

APPENDIX A: MANUAL FOR USE OF THE COMPUTER CODES

A.1 Data Collection

As is often the case with computer techniques, the data requirements seem at first inspection to be much greater than those for making similar decisions subjectively or manually. In one very important sense this is misleading. The user should recognize that for good, but not necessarily optimal, decisions precise data are not required. Reasonable approximations, to a degree of precision equivalent to that which a subjective decision-maker often has intuitively, will usually provide reasonable answers, and the cost of obtaining more precise data may be greater than the savings which result. This idea is of particular importance in estimating the traverse costs of individual streets. The router who does not use computer codes probably uses a mental estimate of traverse cost which is approximately proportional with street length, perhaps modified in occasional cases by his knowledge of particular abnormalities in the streets. Thus, one should be able to obtain results from the computer codes which are of equivalent validity by providing traverse costs which are "eyeballed" from a street map without even very precise measurement. Although it is not recommended that this rather crude approach be used, it is also not recommended that large investments be made in determining actual traverse costs for each and every street in the network. As a compromise, it seems reasonable to divide street segments into several categories, ranging from very slow, winding residential streets to high-speed expressways. Empirical observations of average traverse speed may then be made on several streets of each type, resulting in an estimate of traverse cost per unit distance for each type of street. These costs, coupled with street lengths taken from the best available map, will provide quite satisfactory traverse costs.

For the single-vehicle routing algorithms, no information regarding waste loads on individual streets is required, since the algorithms are applied to known districts which produce one truckload. For the multiple vehicle algorithm, however, an expected load estimate is required for every street segment. Again, it would be pointless to engage in a large-scale data collection study to obtain precise values for these loads. Rather, the region under study may be divided into areas of various population densities, and observations of load per unit distance may be made in each different density area. This will permit making reasonable estimates of expected load for any street from a map, making appropriate adjustments for schools, parks, etc. It is, of course, helpful if these estimates can be made by a person familiar with the area, just as routing without computer algorithms should be done by persons familiar with the area.

Aside from the information discussed above and special data required by some of the algorithms and discussed in the following sections, the only information required for the computer codes is the representation of the street network. This is accomplished by naming the intersections, and identifying the street segments by providing the names of the intersections on which they are incident.

A.2 TOUR1W: A Code for Single-Vehicle Tours on Directed Networks

The input data for TOUR1W consist of:

a) one card which specifies the number of arcs (street segments) in the problem. The format is I4, that is, the number is punched as an integer in columns 1 through 4 of the card.

b) one card for each street segment which provides the following information:

   (1) the name of the intersection at which the street segment starts (i.e., the segment is directed out of this node).

   (2) the name of the intersection at which the street segment ends (i.e., the segment is directed toward this node).

   (3) the traverse cost for the segment

   (4) a 'flag' which indicates that the segment has no collection. This flag is the number '1', punched only for optional street segments, and not punched for segments which must be traversed.

The format for these cards is A4, 1X, A4, 1X, I5, 1X, I1. Thus, the name of the starting node is punched in columns 1 through 4, the name of the ending node is punched in columns 6 through 9, the traverse cost is punched in columns 11 through 15 as an integer, and the flag (if punched) appears in column 17.

No other data are required.

The output from the code first echoes the input data. This is followed by a list of the arcs with the number of times each must be traced in the optimal solution, and one example of an optimal tour, indicated by a list of the intersections in the order visited.

The dimensioned variables in the current implementation will permit the solution of problems consisting of no more than 100 street segments and no more than 100 intersections. In this implementation, the program requires 46,000 bytes on an IBM 360. Larger problems may be solved by changing these dimensions as follows:

a) Dimensioned at number of nodes: NL, NN, NP, IJ, NVISIT

b) Dimensioned at number of nodes + 1: IL, JL

c) Dimensioned at (number of nodes, 10): INC, JINC

d) Dimensioned at number of arcs: JI, KC, KU, KX, JA, IB, LW, SO, SI, ISOURC, ISINK

In addition, NEXTND must be dimensioned sufficiently large to accommodate one element for every appearance of each arc in the optimal tour. Other dimensioned variables in the program should remain unchanged.

TOUR2W: A Code for Single-Vehicle Tours on Undirected Networks

The input data for TOUR2W consist of:

a) one card which specifies the number of edges (street segments) in the problem. The format is I4; that is, the number is punched as an integer in columns 1 through 4 of the card. Dead-end streets should not be included in the input for this program.

b) one card for each street segment, which provides the following information:

(1) the name of one of the intersections on which the segment is incident

(2) the name of the other intersection on which the segment is incident

(3) the traverse cost for the segment

(4) a flag which indicates that the segment has no collection. This flag is the number '1', punched only for optional street segments, and not punched for segments which must be travelled.

The format for these cards is identical to the format for the segment cards in TOUR1W. If the depot is represented, the segment cards for all edges incident on either node of the depot should be input last.

The output from this code provides a list of the edges used in the optimal tour, with retraced edges appearing twice in the list. It then provides a sample tour in which U-turns have not been eliminated, followed by a second tour which has no U-turns.

The dimensioned variables in the current implementation will permit the solution of problems consisting of no more than 270 street segments, and 90 nodes, with no more than 46 nodes of odd degree. In this implementation, the program requires storage of approximately 150,000 bytes on an IBM 360 computer. Large problems may be solved by changing these dimensions as follows:

a) Dimensioned at number of edges: IEND, JEND, COST, IIFLAG, MZ, NZ

b) Dimensioned at number of nodes: NVISIT, IDVEC, JPATH, NODD

c) Dimensioned at (number of nodes, 10): INC

d) Dimensioned at (number of nodes, number of nodes): DIST, IPATH

e) Dimensioned at number of odd nodes: TRND, TREDG, OUTBLS, DISTND, CHANGE, STACK

f) Dimensioned at 1 1/2 times number of odd nodes: NDWT, Z, NDLBL, ROOT, INBLS, BLSND, BLSEDG, STACKA, STACKB, BLSLBL, PARORD, DNODE

g) Dimensioned at (number of odd nodes)x(number of odd nodes + 1)/2: MATCH, II, JJ, I, J, OUTMCH, EDGWT, IINODE, JJNODE, IKM, JKM

In addition, NEXTND and NODAT must be dimensioned sufficiently large to accommodate one element for every appearance of each arc in the optimal tour. The user should also note the sections of COMMON which have been set to dummy variables to retain alignment, and which will have to be changed to correspond with other dimension changes.

TOURM: A Code for Single-Vehicle Tours on Mixed Networks

The input data for TOURM consist of:

a) one card which specifies the number of street segments (both edges and arcs) in the problem. The format is I4. Dead-end streets should not be included in the input for this program.

b) One card for each street segment, which provides the following information:

(1) the name of one of the intersections on which the street segment is incident

(2) the name of the other intersection on which the segment is incident

(3) the traverse cost for the segment

(4) a flag which indicates whether the segment is one-way or two-way. If this flag is the numeral '1', it is assumed that the street segment is one-way, directed from the first named intersection and toward the second names intersection. If the flag is blank, the street segment is assumed to be two-way, and the order of specifying the intersection is immaterial.

The format for these cards is identical to the format for the segment cards in TOUR1W.

The program output first echoes the input data, then provides a list of the number of times each segment is traveled, and finally provides a sample tour.

The current implementation of the program will solve problems with up to 50 intersections, and approximately 70 street segments, depending on the ratio of edges to arcs. In this implementation, approximately 132,000 bytes of storage are required on an IBM 360. Larger problems may be solved by changing dimensions as follows (A=number of arcs, E=number of edges, N=number of nodes):

a) Dimensioned at N: NN, NP, IJ, NL, SUPER, ISET, LABL, LABT, DGRE, LIST, NVISIT

b) Dimensioned at N+1: IL, JL

c) Dimensioned at (N,20): INC, JINC

d) Dimensioned at A+E: ALFA, AS, YS, NODI, NODJ, KOST

e) Dimensioned at 2(A+E): SIGMA

f) Dimensioned at A+2E+2N+1: JI, KC, KU, KX, JA, IB, LW, LARC

g) Dimensioned at (A+E, 2A+2E): BINV

h) Dimensioned at (1.5N, N): NSUPR

In addition, NEXTND should be dimensioned sufficiently large to accommodate one entry for each appearance of each segment in the solution.

MPOST: A Code for Multiple Vehicle Tours on Undirected Networks

The use of MPOST is considerably more complicated than the use of the previous codes. The reader who wishes to use this code is directed to Appendix B of this report (separately bound) which includes a complete description of the MPOST algorithm, a listing of the program, and instructions for preparing input data and interpreting results. The current implementation will solve problems with up to 90 nodes, 270 edges, and 20 districts. Instructions for changing dimensions to solve larger problems are given in Appendix B. The current implementation requires approximately 300,000 bytes of storage on an IBM 360 computer.